

TP368.1/199=2

2007



普通高等教育“十一五”国家级规划教材

高等院校自动化新编系列教材

# 单片机原理与接口技术

(第2版)

编著 马淑华 王凤文 张美金  
主审 崔光照

北京邮电大学出版社  
·北京·

# 高等院校自动化新编系列教材

## 编委会

主 任 汪晋宽

副主任 金海明 罗云林 张美金 崔光照

委 员 (排名不分先后)

于丁文 王凤文 王建国 马淑华 石云霞

齐世清 任彦硕 张家生 张 健 杨建忠

柳明丽 罗长杰 金 伟 赵宏才 赵一丁

顾德英 舒冬梅 藏小杰 郑安平

秘 书 顾德英(兼) 马淑华(兼)

## 编写说明

一本好的教材和一本好的书不同,一本好的书在于其内容的吸引力和情节的魅力,而一本好的教材不仅要对所介绍的科学知识表达清楚、准确,更重要的是在写作手法上能站在读者的立场上,帮助读者对教材的理解,形成知识链条,进而学会举一反三。基于这种考虑,在充分理解自动化专业培养目标和人才需求的前提下,我们规划了这套《高等院校自动化新编系列教材》。

本套系列教材共包括 21 册,在内容取舍划分上,认真分析了各门课程内容的相互关系和衔接,避免了不必要的重复,增加了一些新的内容。在知识结构设计上,保证专业知识完整性的同时,考虑了学生综合能力的培养,并为学生继续学习留有空间。在课程体系规划上,注意了前后知识的贯通,尽可能做到先开的课程为后续的课程提供基础和帮助,后续的课程为先开的课程提供应用的案例,以便于学生对自动化专业的理解。

《高等院校自动化新编系列教材》编委会

2005 年 8 月

# 再版前言

单片微型计算机(Single Chip Microcomputer)简称单片机,是将 CPU、存储器、总线、I/O 接口电路集成在一片超大规模集成电路芯片上,是典型的嵌入式微控制器。单片机的诞生是计算机发展史上的一个新的里程碑。

由于单片机具有体积小、功能全、可靠性好、价格低廉的突出优点,因而问世后广泛应用于工业控制、仪器仪表、交通运输、通信设备、办公设备、家用电器等众多领域,使得许多领域的自动化水平和自动化程度得以大幅度提高,成为现代电子系统中最重要的智能化器件之一。单片机的广泛推广和应用,反过来也进一步使单片机本身得到了迅速的发展,不断地更新换代并逐渐改进和完善各方面的功能。

单片机的典型代表是 Intel 公司在 20 世纪 80 年代初研制出来的 MCS-51 系列单片机,并很快在我国得到广泛的推广和应用。虽然在 90 年代后期美国 Intel 公司把主要精力集中在了 CPU 的研发和生产上,并逐步退出了单片机的市场,但 MCS-51 的核心技术仍然是多家单片机研发和生产公司竞相采用的内核技术。如 ATMEL 公司将其优势的 Flash 技术与 Intel 公司的 80C51 核心技术相结合,生产了 AT89 系列单片机,在继承 MCS-51 单片机的基础上,增加了 Flash 存储器,省去了外部扩展的烦琐,进一步推动了单片机市场的发展。ATMEL 公司的单片机经历了几个发展阶段,从 AT89 系列发展到 AT89C 系列,现在已发展到 AT89S 系列。AT89 系列和 AT89C 系列已渐渐退出市场,被新型的 AT89S 系列单片机所取代。AT89S 系列单片机增加了看门狗(WDT)、SPI、ISP 等技术,性能价格比进一步提高,应用更加方便、可靠。

由于单片机技术的广泛应用性,在我国高等工科院校中,普遍开设了单片机及其相关课程。因此,掌握单片机、应用单片机已成为科研人员必备的技能之一。为了使学校教学与科学技术的飞速发展紧密地联系起来,本书以 AT89S52 单片机为基础,全面介绍了 AT89S52 单片机的基本结构、存储器结构、指令系统、程序设计与调试、I/O 接口、中断、定时器、串行通信以及 AT89S52 单片机的扩展和系统的总体设计。

本书的特点是紧跟单片机技术的发展,注重单片机的实际应用。首先,书中利用一定篇幅对单片机系统的仿真和程序下载进行了介绍,目的是让读者了解单片机开发系统的使用方法,并能够借助于开发系统进行系统的开发。其次是介绍了流行的 C 语言编程方法,并在第 11 章给出了利用 C 语言实现的液氧温度控制系统的编程实例,以期读者在学习汇编语言的时候对单片机的 C 语言编程有一定的了解,为具体的设计和应用打下一定的基础。最后是在相关章节后面附加了实践训练题目,读者可根据书中提供的题目进行相关的实践训练。

本书配套实验板,通过 USB 接口供电,利用串口下载程序,可以完成书中所有的例题



及实践训练项目。

本书是在编委会组织编写人员进行广泛的调研及科学合理的策划,对教材内容及体系结构进行细致认真的审定和推敲,确定编写大纲的基础上,由马淑华、王凤文、张美金具体组织编写工作并担任主编,陈海宴、王军伟、高军参加了部分编写工作。全书由崔光照教授主审,由马淑华统稿。

本书第一版 2005 年正式出版,在我校及其他院校应用,效果较好,并入选教育部组织的“十一五”规划教材。

在本书的编写过程中,还得到了东北大学秦皇岛分校自动化工程系、郑州轻工学院电气信息工程学院、辽宁工程技术大学电气工程系领导的关怀和支持,在此表示感谢。

本书力求与作者的科研经历相结合,但限于编者水平,书中不足之处在所难免,恳请读者批评指正。

编 者

# 目 录

## 第 1 章 单片机概述

1.1 单片机的发展历史 .....	1
1.2 AT89 系列单片机及主要特性 .....	4
1.2.1 低档型 AT89 系列单片机的基本特性 .....	4
1.2.2 标准型 AT89 系列单片机的基本特性 .....	5
1.2.3 高档型 AT89 系列单片机的基本特性 .....	7
1.2.4 AT89 系列单片机型号的编码说明及封装形式 .....	9
1.2.5 部分 ATMEL 单片机的升级替代及推荐产品 .....	11
1.3 单片机的应用 .....	12
1.4 单片机的发展趋势 .....	14

## 第 2 章 AT89S52 单片机的基本结构

2.1 AT89S52 单片机的主要特性 .....	16
2.2 AT89S52 单片机的 CPU .....	17
2.2.1 运算器 .....	17
2.2.2 控制器 .....	19
2.3 存储器和 I/O 接口电路 .....	20
2.4 AT89S52 单片机的封装及引脚功能 .....	20
2.4.1 PDIP 封装的 AT89S52 单片机引脚及功能 .....	21
2.4.2 PLCC 和 TQFP 封装的 AT89S52 单片机引脚及功能 .....	23
2.5 复位操作和复位电路 .....	24
2.6 振荡器、时钟电路及时序 .....	26
2.6.1 振荡器 .....	26
2.6.2 AT89S52 的时序 .....	27
2.7 AT89S52 的低功耗工作方式 .....	29
习题 .....	31

## 第 3 章 AT89S52 存储器结构

3.1 存储器概述 .....	33
3.2 AT89S52 单片机的存储器结构 .....	34

3.2.1 程序存储器·····	34
3.2.2 数据存储器·····	35
3.2.3 特殊功能寄存器 SFR ·····	37
3.3 外部存储器及其访问·····	40
3.3.1 外部程序存储器及访问·····	40
3.3.2 外部数据存储器及访问·····	43
3.4 片内 Flash 存储器操作 ·····	45
3.4.1 标志字节·····	45
3.4.2 程序存储器的加密·····	45
3.4.3 Flash 存储器的并行编程 ·····	46
3.4.4 Flash 存储器的串行编程 ·····	48
习题 ·····	51

## 第 4 章 AT89S52 指令系统

4.1 汇编语言指令格式·····	54
4.1.1 汇编语言执行指令格式·····	54
4.1.2 汇编伪指令·····	54
4.2 寻址方式·····	57
4.3 指令系统·····	62
4.3.1 数据传送指令·····	62
4.3.2 算术运算指令·····	67
4.3.3 逻辑运算指令·····	72
4.3.4 位(布尔)操作类指令·····	74
4.3.5 控制转移类指令·····	77
习题 ·····	84

## 第 5 章 AT89S52 程序设计与调试

5.1 程序设计步骤·····	87
5.2 源程序的基本格式及编辑环境·····	89
5.2.1 源程序的基本格式·····	89
5.2.2 源程序的编辑环境·····	91
5.3 程序设计方法·····	91
5.3.1 顺序结构程序·····	91
5.3.2 分支结构程序·····	92
5.3.3 循环结构程序·····	94
5.3.4 子程序结构程序·····	97
5.3.5 中断服务程序 ·····	102
5.4 C51 基础 ·····	103

5.4.1 C51 的程序结构及编译环境 .....	103
5.4.2 C51 的数据类型与存储类型 .....	105
5.4.3 AT89S52 结构的 C51 定义 .....	108
5.4.4 C51 和汇编语言的混合编程 .....	111
5.4.5 C51 程序设计举例 .....	117
5.5 程序调试与下载运行 .....	119
5.5.1 单片机开发系统 .....	120
5.5.2 源程序调试 .....	121
5.5.3 程序下载运行 .....	122
习题 .....	123
实践训练 .....	123

## 第 6 章 AT89S52 单片机并行 I/O 接口

6.1 I/O 接口概述 .....	125
6.1.1 I/O 接口的功能 .....	125
6.1.2 接口与端口 .....	126
6.1.3 I/O 接口编址技术 .....	127
6.1.4 I/O 数据传送的控制方式 .....	127
6.2 AT89S52 并行 I/O 接口的内部结构 .....	130
6.2.1 I/O 接口的结构特点 .....	130
6.2.2 AT89S52 的并行 I/O 接口 .....	131
6.3 并行 I/O 接口操作 .....	136
6.4 I/O 接口应用 .....	137
6.4.1 I/O 接口应用特性 .....	137
6.4.2 I/O 接口的应用 .....	138
习题 .....	144
实践训练 .....	145

## 第 7 章 AT89S52 单片机中断系统

7.1 中断概述 .....	146
7.2 中断系统结构与中断控制 .....	148
7.2.1 AT89S52 的中断源 .....	148
7.2.2 中断标志与控制 .....	151
7.3 中断响应 .....	154
7.3.1 中断响应条件 .....	154
7.3.2 中断响应过程 .....	155
7.3.3 中断响应时间 .....	156
7.4 中断请求的撤除 .....	157

7.5 外部中断源的扩展 .....	158
7.5.1 采用“OC 门”经“线或”扩展中断源 .....	158
7.5.2 通过片内定时/计数器扩展中断源 .....	160
7.6 中断程序设计 .....	160
习题 .....	165
实践训练 .....	165

## 第 8 章 AT89S52 定时/计数器

8.1 定时/计数器 0/1 的结构 .....	167
8.2 定时/计数器 0/1 的控制 .....	168
8.2.1 定时/计数器 0/1 工作模式寄存器 TMOD .....	168
8.2.2 定时/计数器 0/1 控制寄存器 TCON .....	169
8.3 定时/计数器 0/1 的 4 种模式及应用 .....	169
8.3.1 模式 0 及应用 .....	170
8.3.2 模式 1 及应用 .....	172
8.3.3 模式 2 及应用 .....	173
8.3.4 模式 3 及应用 .....	175
8.3.5 定时/计数器的其他应用 .....	176
8.4 定时/计数器 T2 .....	177
8.4.1 T2 控制寄存器 .....	177
8.4.2 T2 模式寄存器 .....	178
8.4.3 T2 的工作模式 .....	178
8.5 定时监视器 .....	182
8.5.1 AT89S52 的定时监视器 .....	182
8.5.2 辅助功能寄存器 AUXR .....	183
习题 .....	184
实践训练 .....	184

## 第 9 章 AT89S52 单片机串行通信

9.1 串行通信概述 .....	185
9.1.1 串行通信的实现 .....	185
9.1.2 串行通信的通信方式 .....	186
9.1.3 串行通信的传输方式 .....	188
9.2 RS232C 标准总线及通信设计 .....	189
9.2.1 RS232C 接口的引脚描述 .....	189
9.2.2 RS232C 接口的具体规定 .....	190
9.2.3 RS232C 接口的典型应用 .....	191
9.3 AT89S52 串行通信接口 .....	193

9.3.1 串行口的控制 .....	193
9.3.2 串行口的工作模式 .....	195
9.3.3 多机通信 .....	200
9.3.4 波特率的确定 .....	201
9.4 串行通信应用举例 .....	203
9.4.1 串行口模式 0 的应用 .....	203
9.4.2 串行口模式 1 的应用 .....	204
9.4.3 串行口模式 2 和模式 3 的应用 .....	206
习题 .....	209
实践训练 .....	209

## 第 10 章 单片机应用系统扩展技术

10.1 总线扩展及地址分配 .....	211
10.1.1 总线扩展 .....	211
10.1.2 地址分配 .....	213
10.2 外部程序存储器扩展 .....	215
10.2.1 常用 EPROM 芯片 .....	215
10.2.2 典型 EPROM 扩展电路实现 .....	217
10.3 外部数据存储器的扩展 .....	218
10.3.1 RAM(SRAM)的扩展 .....	218
10.3.2 并行 EEPROM 的扩展 .....	220
10.3.3 串行 EEPROM 的扩展 .....	222
10.4 并行 I/O 接口的扩展 .....	227
10.4.1 简单 I/O 接口的扩展 .....	227
10.4.2 可编程 8155 的并行 I/O 扩展 .....	228
10.4.3 8255A 可编程并行 I/O 接口扩展 .....	239
10.5 A/D 和 D/A 转换接口的扩展 .....	246
10.5.1 8 位并行 A/D 转换器 ADC0809 的扩展 .....	247
10.5.2 12 位并行 A/D 转换器 AD574 的扩展 .....	250
10.5.3 12 位串行 A/D 转换器 TLC2543 的扩展 .....	252
10.5.4 8 位并行 D/A 转换器 DAC0832 的扩展 .....	255
10.5.5 12 位串行 D/A 转换器 TLV5616 的扩展 .....	259
10.6 实时时钟电路 DS1302 的扩展 .....	263
习题 .....	268
实践训练 .....	268

## 第 11 章 单片机应用系统设计及举例

11.1 单片机应用系统的开发过程 .....	270
-------------------------	-----

11.2 液氧容器温度控制系统设计·····	273
11.2.1 系统的目标任务·····	273
11.2.2 系统的总体设计·····	273
11.2.3 系统的结构框图及工作原理·····	274
11.2.4 硬件设计·····	275
11.2.5 软件设计·····	277
11.3 基于 GSM/CDMA 的防盗报警系统 ·····	282
11.3.1 系统的目标任务·····	282
11.3.2 系统的总体设计·····	282
11.3.3 系统的结构框图及工作原理·····	283
11.3.4 程序流程及软件设计 ·····	283
习题·····	289
实践训练·····	290

## 第 12 章 其他系列单片机介绍

12.1 HOLTEK 公司的 HT48××系列单片机概述 ·····	291
12.1.1 HT48××系列单片机的主要性能 ·····	291
12.1.2 HT48××系列单片机的引脚描述 ·····	292
12.1.3 HT48××系列单片机的内部结构框图 ·····	292
12.1.4 HT48××系列单片机的指令集 ·····	293
12.2 PIC16C5×系列单片机概述 ·····	296
12.2.1 PIC16C5×系列单片机的主要性能 ·····	296
12.2.2 PIC16C5×系列单片机的引脚描述 ·····	297
12.2.3 PIC16C5×系列单片机的内部结构框图 ·····	298
12.2.4 PIC16C5×系列单片机的指令集 ·····	299
12.3 其他型号单片机及其生产厂商简介·····	300

附录 AT89S52 单片机实验系统 ·····	303
--------------------------	-----

参考文献·····	314
-----------	-----



# 第 1 章 单片机概述

进入 20 世纪 60 年代,世界在大规模和超大规模集成电路的制造水平和工艺上取得了飞速的进步。1971 年,美国的 Intel 公司研究并制造了 140004 微处理器芯片。该微处理器是将以往分立的运算器、控制器和寄存器集成在一块芯片上,因此又称为中央处理单元(CPU)。140004 微处理器芯片是世界上第一个微处理器芯片,以它为核心组成的 MCS-48 计算机是世界上第一台微型计算机。微型计算机就是以微处理器为核心,采用系统总线技术,配以采用了大规模或超大规模集成电路的存储器、I/O 接口电路、I/O 设备所组成的计算机。

单片机(Single Chip Microcomputer 或 One Chip Microcomputer)的全称为单片微型计算机,是微型计算机家族中的一个分类,是将 CPU、存储器、总线、I/O 接口电路集成在一片超大规模集成电路芯片上。单片机具有体积小、功能全、价格低廉的突出优点,同时其软件也非常丰富,并可将这些软件嵌入到其他产品中,使其他产品具有丰富的智能。单片机所具有的这些优点使之问世后得到了迅速的发展,广泛应用在工业控制、仪器仪表、交通运输、通信设备、办公设备、家用电器等众多领域,成为现代电子系统中最重要的智能化器件。

## 1.1 单片机的发展历史

单片机的发展经历了 4 个阶段:初级阶段、技术成熟阶段、发展和推广阶段及 16 位单片机阶段。

### 1. 第一阶段

1974—1976 年,是单片机的初级阶段。

这一阶段单片机的主要特点是功能和结构都比较简单,芯片内只包含了 8 位的 CPU、64 B 的随机读写数据存储器(RAM)和 2 个并行输入/输出(I/O)接口。并且由于受制造水平和工艺的限制,芯片采用了双片结构,还需要外接一个内含 ROM、定时/计数器和并行 I/O 接口电路的芯片才能构成一台完整的单片微型计算机,还没有形成真正意义上的单片机。

### 2. 第二阶段

1976—1980 年,是单片机技术走向成熟的阶段。

这一阶段的单片机在性能和结构上有所提高和改进,但其性能仍然比较低,因此也将这一阶段的单片机称为低性能单片机阶段。

虽然这一阶段单片机的性能仍然比较低,但随着超大规模集成电路制造水平和工艺的进步,形成了真正的单片结构。这一阶段的典型代表是美国 Intel 公司于 1976 年推出

的 MCS-48 系列单片机,这是第一代通用的单片机。这一通用系列单片机的推出,开辟了单片机的市场,促进了单片机技术的迅猛发展和进步。这一系列单片机的基本型产品为 8048,其内含 8 位的 CPU、64 B 的 RAM 数据存储器、1 KB 的 ROM 程序存储器、一个 8 位的定时/计数器和 27 根 I/O 口线,表 1.1.1 列出了 MCS-48 系列单片机的型号和性能。从表中可以看到,其 P8748H 和 P8749H 是片内 ROM 采用了 EPROM 形式的 8048AH 和 8049AH,从这一阶段开始可以方便地改写控制程序。

表 1.1.1 MCS-48 系列单片机的型号和性能

型 号	CPU	ROM	RAM	定时/计数器	I/O 口线
8035AHL	8 位	无	64 B	1×8 位	15
8039AHL	8 位	无	128 B	1×8 位	15
8040AHL	8 位	无	256 B	1×8 位	15
8048AH	8 位	1 KB	64 B	1×8 位	27
8049AH	8 位	2 KB	128 B	1×8 位	27
8050AH	8 位	4 KB	256 B	1×8 位	27
P8748H	8 位	1 KB EPROM	64 B	1×8 位	27
P8749H	8 位	2 KB EPROM	128 B	1×8 位	27

### 3. 第三阶段

1980—1983 年,是单片机技术的发展和推广阶段。

进入 20 世纪 70 年代末 80 年代初,在超大规模集成电路制造水平和工艺得到迅猛发展的同时,微处理器技术也得以迅速发展,在这一阶段单片机技术更加成熟。

这一阶段单片机性能有了很大的提高,虽然 CPU 仍然是 8 位,但频率已经提高到了 12 MHz。芯片内 ROM 最多达到 8 KB,并开始普遍应用 EPROM,寻址范围达到了 64 KB,芯片内 RAM 的数量最少也达到了 128 B,I/O 口线的数量也达到了 32 位,因此又将这一阶段称为高性能单片机阶段。

进入 70 年代后期,许多半导体公司看到了单片机巨大的市场前景,纷纷加入到这一领域的开发研制之中,推出了多个品种的系列机。这一阶段的典型代表是 Intel 公司于 1980 年推出的 MCS-51 系列单片机,表 1.1.2 给出了 MCS-51 系列单片机部分产品的型号和性能。

表 1.1.2 MCS-51 系列单片机的型号和性能

型 号		CPU	ROM	RAM	定时/计数器	I/O 口线
8051	8031AH	8 位	无	128 B	2×16 位	32
	8051AH	8 位	4 KB	128 B	2×16 位	32
	8051BH	8 位	4 KB	128 B	2×16 位	32
	8751AH	8 位	4 KB EPROM	128 B	2×16 位	32
	8751BH	8 位	4 KB EPROM	128 B	2×16 位	32

续表

型 号		CPU	ROM	RAM	定时/计数器	I/O 口线
8052	8032BH	8 位	无	256 B	3×16 位	32
	8052BH	8 位	8 KB ROM	256 B	3×16 位	32
	8752BH	8 位	8 KB EPROM	256 B	3×16 位	32
80C51	80C31BH	8 位	无	128 B	2×16 位	32
	80C51BH	8 位	4 KB ROM	128 B	2×16 位	32
	80C51BHP	8 位	4 KB ROM	128 B	2×16 位	32
	87C51	8 位	4 KB EPROM	128 B	2×16 位	32
	83C51FA	8 位	8 KB ROM	256 B	3×16 位	32
	87C51FA	8 位	8 KB EPROM	256 B	3×16 位	32

从表中可以看到,8031 芯片内没有 ROM,使用时需要外接 EPROM 芯片,其他与 8051 完全相同,8051AH 和 8051BH 的区别是可以对 8051BH 芯片中 ROM 内的程序进行加密,防止被他人改写或抄袭。8751 是芯片内采用了 EPROM 的 8051。8751AH 和 8751BH 的区别是 8751BH 芯片中设有二级保密位,而 8751AH 芯片中只设有一级保密位。8051 和 80C51 的区别是 8051 采用 HMOS 工艺制造,而 80C51 采用 CHMOS 工艺制造,CHMOS 工艺技术先进,它同时具有 HMOS 的高速度和 CMOS 的低功耗的优点,除制造工艺的区别外,其他均兼容。

8052 是 8051 的增强型,除与 8051 完全兼容外,还增加了 128 B 的片内 RAM、4 KB 的 ROM 或 EPROM、1 个定时/计数器、1 个中断源。

对比表 1.1.1 和表 1.1.2 不难看到,代表着单片机两个发展阶段的典型产品在性能方面都有了哪些提高。

虽然在 20 世纪 90 年代后期,美国 Intel 公司出于公司发展战略的考虑将主要精力集中在了 CPU 的研发和生产上,并逐步退出了单片机的市场,但 MCS-51 的核心技术仍然是多家单片机研发和生产公司竞相采用的内核技术。

#### 4. 第四阶段

1983 年到现在,这一阶段单片机技术的发展主要体现在内部资源的增加和实时处理功能的加强方面,增加了多通道 10 位的 A/D 转换器、高速输入/输出部件(HSIO)、脉宽调制输出装置(PWM)、外围传送服务功能等,CPU 的位数达到了 16 位、32 位,因此又称这一阶段为 16 位单片机阶段。这一阶段的典型代表是 Intel 公司于 1983 年推出的 MCS-96 系列单片机,表 1.1.3 为 MCS-96 系列单片机的部分产品的型号和主要性能。

表 1.1.3 MCS-96 系列单片机部分产品的型号和性能

型 号	ROM	RAM	A/D	HSIO/EPA	PWM	定时/计数器	I/O 口线
8398	8 KB	232 B	4×10	HSIO	1	2×16 位	48
8397BH	8 KB	232 B	8×10	HSIO	1	2×16 位	68
8397JF	16 KB	488 B	8×10	HSIO	1	2×16 位	68
83C198	8 KB	232 B	4×10	HSIO	1	2×16 位	52
83C196KB	8 KB	232 B	8×10	HSIO	1	2×16 位	68
83C196KC	16 KB	488 B	8×10	HSIO	3	2×16 位	68
83C196KR	16 KB	744 B	8×10	EPA	—	2×16 位	68
83C196MC	16 KB	488 B	13×10	EPA	—	2×16 位	84

## 1.2 AT89 系列单片机及主要特性

由 1.1 节中已经了解到,在 20 世纪 90 年代后期美国 Intel 公司出于公司发展战略的考虑将主要精力集中在了 CPU 的研发和生产上,并逐步退出了单片机的市场,但在单片机的发展和应用历史中,MCS-51 系列单片机已经得到科技界和工业界广大用户最广泛的认可。虽然许多半导体公司看到了单片机巨大的市场前景并纷纷加入到这一领域的开发研制,并为满足各种不同的需求推出了多个品种的系列机,这些单片机产品还采用了多种创新技术,产品的性能和可靠性都有了极大的改进和提高,但这些单片机产品大都采用了 8051 的核心技术作为其内核,例如美国 ATMEL 公司研发的 AT89 系列、ADI 公司的 AD $\mu$ C 系列、Philips 公司研发的 80C51 系列、Motorola 公司推出的 M68HC05 系列等。

AT89 系列单片机是美国 ATMEL 公司的产品。ATMEL 公司成立于 20 世纪 80 年代中期,公司成立后将研发方向定位为新型半导体存储技术,并很快取得了成效,在 Flash 存储器技术领域取得了优势,并创造性地将 Flash 存储器技术注入到单片机产品中,将 Flash 存储器技术与 Intel 公司的 MCS-51 的核心技术相结合,在 20 世纪末推出了 AT89 多种系列单片机。

AT89 系列单片机的内部功能、引脚的数量和排列方式、指令系统与 MCS-51 系列单片机完全兼容,因此对于以 MCS-51 系列产品为基础的应用系统而言,十分容易进行替换。AT89 单片机拥有着较庞大的家族系列,每一系列下都有多个型号,而每个型号还有多个具体的型号。AT89 多种系列单片机可分为低档型、标准型和高档型 3 个系列,下面对这 3 个系列的 AT89 单片机的主要性能进行系统的介绍,以求从整体上对 AT89 系列单片机有一个了解。

### 1.2.1 低档型 AT89 系列单片机的基本特性

低档型 AT89 系列单片机是 AT89 多种系列单片机中的低档型产品。

所谓低档型指的是在标准型的结构基础上,为了适应一些简单的控制系统的需要而适当地减少一些功能部件,形成一种体积更加小巧、功能简化或单一、价格更加低廉的单

片机。本小节中对 AT89C1051 系列中的 AT89C1051U、AT89C2051 和 AT89C4051 单片机的基本特性作一简单的介绍和比较。

### 1. AT89C1051U 单片机的基本特性

- 8031CPU;
- 1 KB 的快速擦写 Flash 存储器,用于程序存储,可擦写次数为 1 000 次;
- 芯片内数据存储器空间包括 64 B 的芯片内 RAM(00H~3FH)和 128 B 的特殊功能寄存器 SFR 区域(80H~FFH);
- 15 条可编程 I/O 口线;
- 2 个可编程 16 位定时器;
- 具有 6 个中断源、5 个中断矢量、2 级优先权的中断系统;
- 1 个可编程的 UART(Universal Asynchronous Receiver and Transmitter,通用异步收发器)串行通信口;
- 具有“空闲”和“掉电”两种低功耗工作方式;
- 可编程的 2 级程序锁定位;
- 工作电源的电压为 2.7~6.0 V;
- 完全静态操作模式为 0~24 MHz;
- 内部含有一个模拟比较器;
- 可直接驱动 LED。

AT89C1051U 单片机的引脚排列如图

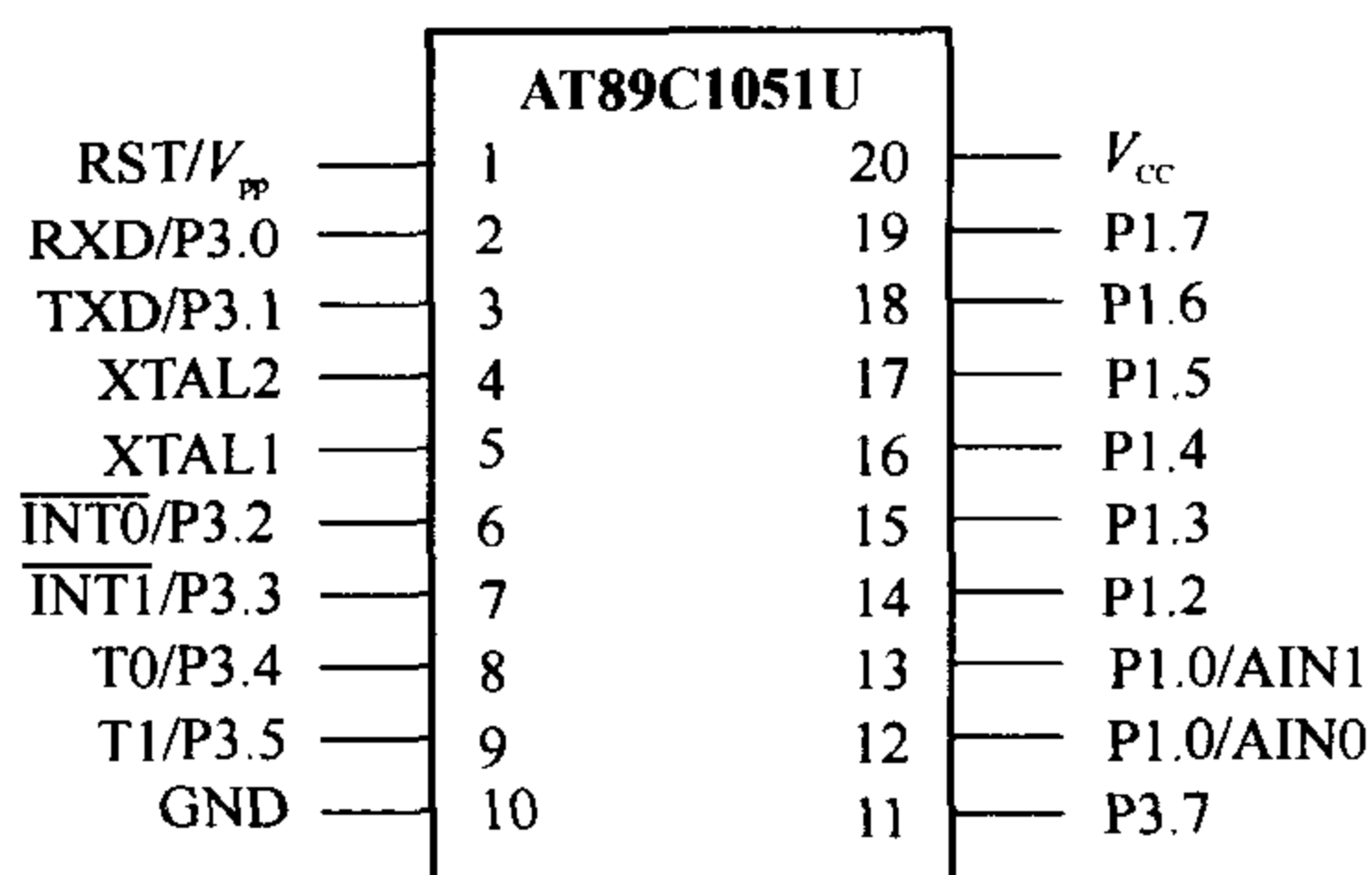


图 1.2.1 AT89C1051U 单片机的引脚排列

1.2.1 所示。

### 2. AT89C2051 单片机的基本特性

AT89C2051 单片机的引脚排列与 AT89C1051U 单片机的引脚排列完全相同。与 AT89C1051U 单片机相比,AT89C2051 增加了 64 B 的片内 RAM。

### 3. AT89C4051 单片机的基本特性

AT89C4051 单片机的引脚排列与 AT89C1051U 单片机的引脚排列完全相同。与 AT89C2051 相比,AT89C4051C 除增加了 2 KB 的片内 ROM 外,还增加了电压掉电检测功能。所谓电压掉电检测就是当电源电压  $V_{cc}$  下降到检测门限电压以下,并重新升高超过检测门限电压延迟 15 ms 后,单片机内部将自动产生一个复位信号。

## 1.2.2 标准型 AT89 系列单片机的基本特性

标准型 AT89 系列单片机包括 AT89C51、AT89C52、AT89S51 和 AT89S52。由于标准型 AT89 系列单片机与 MCS-51 完全兼容,又有着优良的特性以及较高的性能价格比,因此成为 AT89 多种系列单片机家族中的主流机型。在标准型 AT89 单片机的基础上适当减少或增加部分硬件,则可方便地形成低档型 AT89 系列单片机或高档型 AT89 系列单片机。

## 1. AT89C51 的主要工作特性

- 8031CPU;
- 4 KB 的快速擦写 Flash 存储器,用于程序存储,可擦写次数为 1 000 次;
- 256 B 的 RAM,其中高 128 B 字节地址被特殊功能寄存器 SFR 占用;
- 32 条可编程 I/O 口线;
- 2 个可编程 16 位定时器;
- 具有 6 个中断源、5 个中断矢量、2 级优先权的中断系统;
- 一个数据指针 DPTR;
- 1 个可编程的全双工串行通信口;
- 具有“空闲”和“掉电”两种低功耗工作方式;
- 可编程的 3 级程序锁定位;
- 工作电源的电压为  $(5 \pm 0.2) \text{V}$ ;
- 振荡器最高频率为 24 MHz;
- 编程频率 3~24 MHz,编程电流 1 mA,编程电压  $V_{\text{PP}}$  为 5 V 或 12 V。

PDIP 封装形式的 AT89C51 单片机的引脚排列如图 1.2.2 所示。

从图 1.2.2 看到,从低档型 AT89 系列单片机进入到标准型 AT89 系列单片机后其引脚数由 20 个增加到了 40 个,其中的 I/O 口线由 15 条增加到了 32 根,并增加了多个控制信号,串行通信口由通用异步收发器变成全双工,程序锁定位由 2 级增加到 3 级。

## 2. AT89C52 的主要工作特性

AT89C52 单片机的引脚排列除 P1.0 口和 P1.1 口与 AT89C51 有所不同外,其他均相同,如图 1.2.3 所示。

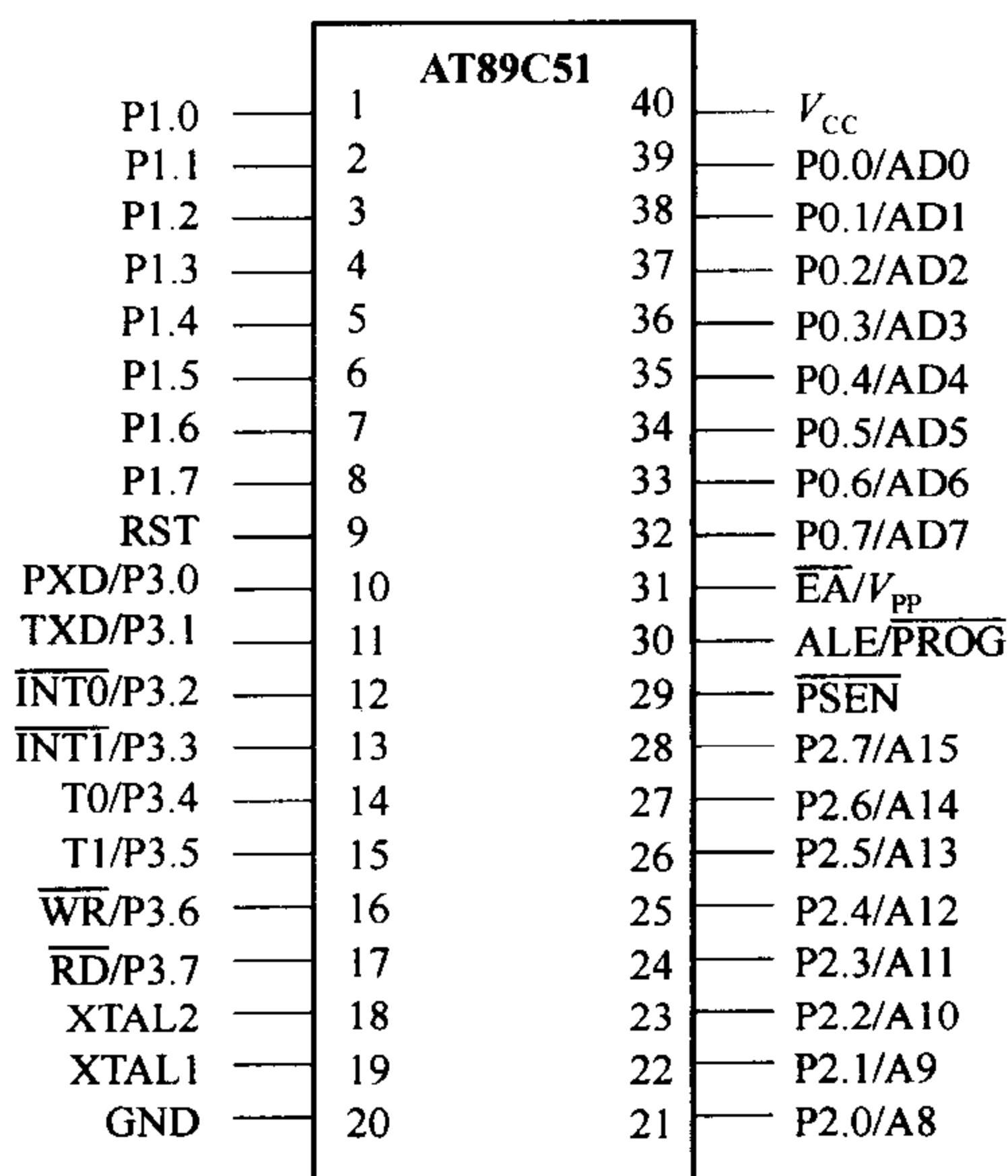


图 1.2.2 PDIP 封装形式的 AT89C51 单片机  
引脚排列

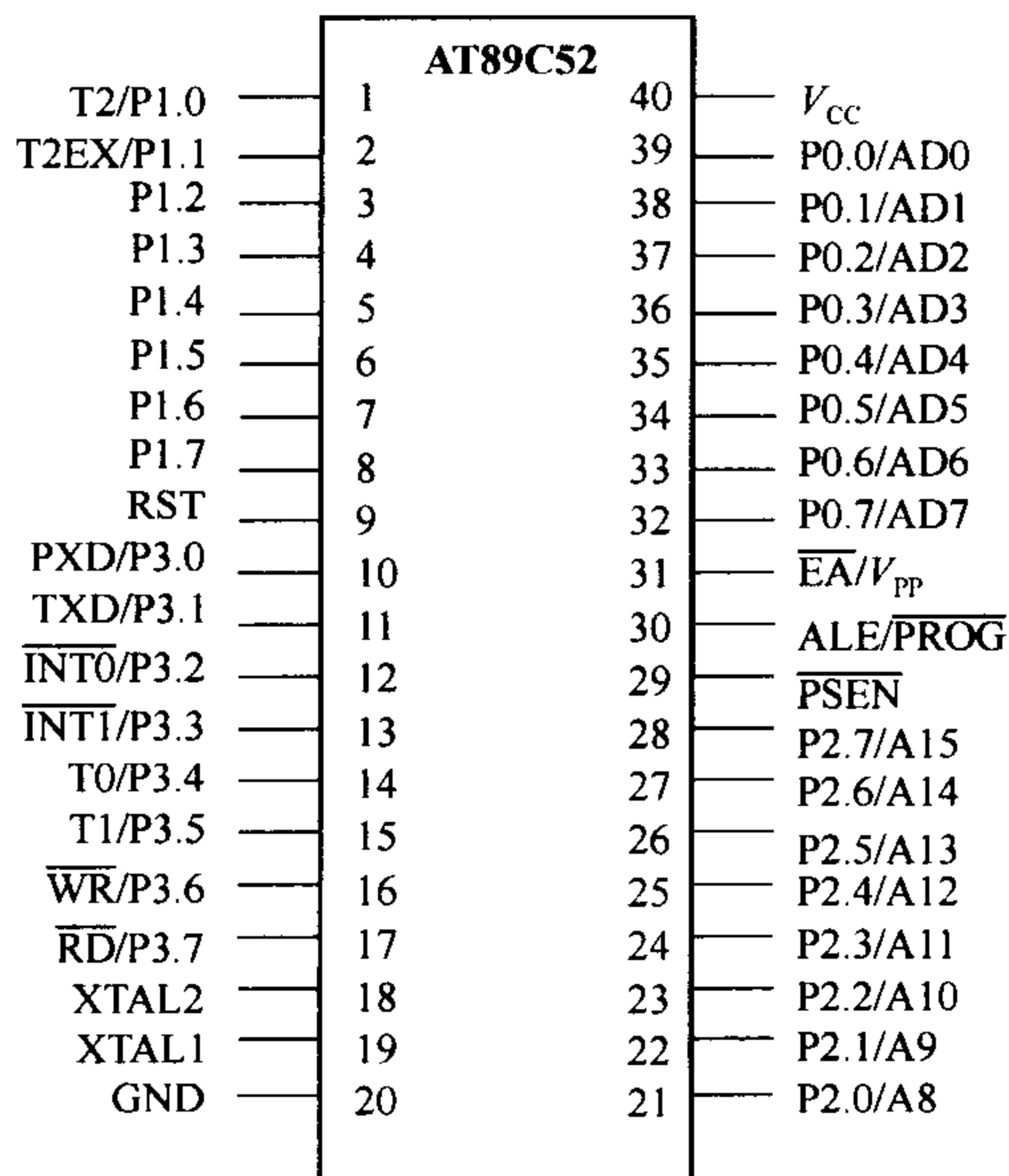


图 1.2.3 PDIP 封装形式的 AT89C52 单片机  
引脚排列

### 3. AT89S51 的主要工作特性

AT89S51 单片机是一种低功耗、具有在线编程 Flash 程序存储器的单片机。所谓在线编程(ISP, In System Program)指的是允许单片机芯片在不离开电路板或不离开设备的情况下,实现固化和擦除操作,在线编程给单片机用户的研发和使用带来了极大的方便。AT89S51 与 AT89C51 单片机的工作特性相比较,主要增加了以下功能:

- 增加了在线编程功能,使程序的修改和调试极其方便,而且编程和校验也更加方便、灵活;
- 数据指针由 1 个增加到 2 个,使对扩展外部数据存储器的访问更加方便;
- 增加了看门狗定时器 WDT,使单片机应用系统的抗干扰能力得到提高;
- 增加了断电标志 POF;
- 增加了掉电状态下的中断恢复方式。

AT89S51 单片机的引脚排列与 AT89C51 的引脚排列基本相同,只是在 6、7、8 引脚增加了串行编程和校验时的串行数据输入、输出和移位脉冲输入功能。

### 4. AT89S52 的主要工作特性

与 AT89S51 单片机相比,AT89S52 单片机主要增加了以下的功能特性:

- 芯片内的 Flash 程序存储器由 4 KB 增加到 8 KB;
- 芯片内的数据存储器由 128 B 增加到 256 B;
- 芯片内新增加了一个定时器 T2,芯片内定时器总数增加到 3 个(T0、T1 和 T2);
- 中断源由原来的 6 个增加到 8 个,中断矢量由 5 个增加到 6 个。

AT89S52 单片机的引脚排列与 AT89S51 的引脚排列基本相同,只是在引脚 1 (P1.0)和引脚 2(P1.1)增加了定时器 2 的外部计数输入和触发器输入。AT89S52 单片机的引脚排列详见 2.4 节。

## 1.2.3 高档型 AT89 系列单片机的基本特性

所谓高档型单片机是指在标准型单片机结构的基础上,增加一部分功能部件,使之具备比标准型单片机更高、更优良的性能。

高档型 AT89 系列单片机包括了 AT89C51RC、AT89S8252、AT89S53 和 AT89C55WD 等。

### 1. AT89C51RC 单片机

AT89C51RC 单片机是在 AT89C52 基础上开发的高档型单片机,其主要工作特性如下:

- 8031CPU;
- 32 KB 的 Flash 程序存储器,可擦写次数为 1 000 次;
- 512 B 的片内数据存储器 RAM(不包括 128 B 的特殊功能寄存器 SFR);
- 32 条可编程 I/O 口线(P0~P3);
- 3 个可编程 16 位定时器 T0、T1 和 T2;
- 具有 8 个中断源、6 个中断矢量、2 级优先权的中断系统;
- 双数据指针 DPTR0 和 DPTR1;



- 1 个可编程的全双工串行通信口;
- 1 个看门狗定时器 WDT;
- 具有“空闲”和“掉电”两种低功耗工作方式;
- 可编程的 3 级程序锁定位;
- 断电标志 POF;
- 工作电源的电压为 4.0~5.5 V;
- 振荡器最高频率为 33 MHz。

与 AT89C52 单片机相比,AT89C51RC 单片机主要增加了以下的功能特性:

- 芯片内的 Flash 程序存储器由 8 KB 增加到 32 KB;
- 芯片内的数据存储器由 256 B 增加到 512 B;
- 数据指针由 1 个增加到 2 个;
- 增加了看门狗定时器 WDT;
- 退出掉电工作方式时,由 AT89C52 单片机的单纯硬件复位增加了中断响应后复位的功能;
- 增加了断电标志 POF。

AT89C51RC 单片机的引脚排列与 AT89C52 的引脚排列完全相同。

## 2. AT89S8252 单片机

与前面介绍的各种 AT89 系列单片机不同,AT89S8252 单片机除 Flash 程序存储器外还增加了可擦写 10 万次的 2 KB EEPROM 存储器,中断系统增加到了 9 个中断源,具有 SPI(Serial Peripheral Bus)串行总线接口,其主要工作特性如下:

- 8031CPU;
- 8 KB 的快速擦写 Flash 程序存储器,可擦写次数为 1 000 次;
- 2 KB 的 EEPROM 程序存储器,可擦写 10 万次;
- 256 B 的片内数据存储器 RAM(不包括 128 B 的特殊功能寄存器 SFR);
- 32 条可编程 I/O 口线;
- 3 个可编程 16 位定时器;
- 具有 9 个中断源、6 个中断矢量、2 级优先权的中断系统;
- 双数据指针 DPTR0 和 DPTR1;
- 1 个可编程的 UART 串行通信口;
- 具有“空闲”和“掉电”两种低功耗工作模式;
- 可编程的 3 级程序锁定位;
- 断电标志 POF;
- SPI 外围扩展串行口;
- 工作电源的电压为 4.0~6.0 V;
- 振荡器最高频率为 24 MHz。

AT89S8252 单片机的引脚排列与 AT89S52 的引脚排列基本相同,只是在引脚

5(P1.4)新增加了从器件选择线 $\overline{SS}$ 的功能。

### 3. AT89S53 单片机

AT89S53 单片机是在 AT89C52 基础上开发的增强型产品,与 AT89C52 相比增加了如下功能:

- 芯片内的 Flash 程序存储器由 8 KB 增加到 12 KB;
- 新增加了 SPI 外围扩展串行口;
- 对 Flash 程序存储器可使用串行口进行编程和校验;
- 数据指针由 1 个增加到 2 个;
- 增加了看门狗定时器 WDT;
- 退出掉电工作方式时可采用外部中断方式;
- 中断源由 8 个增加到 9 个;
- 具有断电标志 POF。

AT89S53 单片机的引脚排列与 AT89S8252 的引脚排列完全相同。

### 4. AT89C55WD 单片机

AT89C55WD 单片机也属于 AT89C52 的增强型产品,与 AT89C52 相比增加了如下功能:

- 芯片内的 Flash 程序存储器由 8 KB 增加到 20 KB;
- 新增加了 SPI 外围扩展串行口;
- 最高工作频率由 AT89C52 的 24 MHz 提高到 33 MHz;
- 数据指针增加到 2 个;
- 增加了看门狗定时器 WDT;
- 退出掉电工作方式时可采用外部中断方式;
- 增加了断电标志 POF。

AT89C55WD 单片机的引脚排列与 AT89SC52 的引脚排列完全相同。

## 1.2.4 AT89 系列单片机型号的编码说明及封装形式

### 1. 编码说明

AT89 系列单片机型号的编码由前缀、型号和后缀 3 部分组成,格式如表 1.2.1 所示。

表 1.2.1 AT89 系列单片机型号的编码

前缀	型号			分隔符	后缀
AT	89	C	×××× (最多 4 位)	—	××××
		LV			
		S			

前缀 AT 表示该产品由美国 ATMEL 公司生产。型号又分为 3 部分:其中 89 的 9 表

示单片机内含 Flash 存储器;第二部分中的 C 代表产品采用 CMOS 技术生产, LV 表示产品为低压产品, S 表示该型号的产品支持在线编程;型号的最后部分最多 4 位, 表示产品的具体型号, 如 51、52、2051 等。

型号编码的后缀由 4 个参数组成, 每个参数又有不同的参数值代表不同的意义, 如表 1.2.2 所示。

表 1.2.2 AT89 系列单片机型号的后缀说明

位	内 容	含 义
第 1 位 代表可以支持的最高 系统时钟频率	12	振荡频率最高为 12 MHz
	16	振荡频率最高为 16 MHz
	20	振荡频率最高为 20 MHz
	24	振荡频率最高为 24 MHz
第 2 位 代表封装形式	D	CERDIP
	J	表示塑料芯片载体, PLCC 封装
	L	表示陶瓷芯片载体, LCC 封装
	P	表示塑料双列直插 PDIP 形式封装
	S	表示用 SOIC 形式封装
	Q	表示用 PQFP 形式封装
	A	表示用 TQFP 形式封装
第 3 位 代表应用极别	C	表示商业用产品, 温度范围 $0\sim+70^{\circ}\text{C}$
	I <sup>①</sup>	表示工业用产品, 温度范围 $-40\sim+85^{\circ}\text{C}$
	A	表示汽车用产品, 温度范围 $-40\sim+125^{\circ}\text{C}$
	M	表示军用产品, 温度范围 $-55\sim+150^{\circ}\text{C}$
第 4 位	空	处理工艺为标准工艺
	/813	处理工艺采用 MIL-STD-883 标准
	L	表示无引线芯片载体

① 需要说明的是, 由于欧美要求使用无铅 IC, 所以 ATMEL 公司将推出带“U”的单片机, 取代原来带“I”的型号。  
如 AT89S52-24AU 将取代 AT89S52-24AI。

例如某单片机的型号为 AT89C52-20AC, 该型号代表的含义: ATMEL 公司生产的含有 Flash 存储器的单片机, 采用 CMOS 技术生产, 内部为 51 结构, 频率为 20 MHz, 采用 TQFP 形式封装, 商业用产品, 温度范围  $0\sim+70^{\circ}\text{C}$ 。

## 2. 单片机的封装形式

单片机的封装形式有 PDIP、TQFP、PLCC 等多种形式, 各种封装形式的说明如下。

PDIP (Plastic Dual Inline Package)——塑封双列直插式封装, 可直接插入标准插座或焊在印制板上。

PQFP (Plastic Quad Flat Package)——塑封方形贴片式封装, 可直接将引脚敷贴在印制板上焊牢。

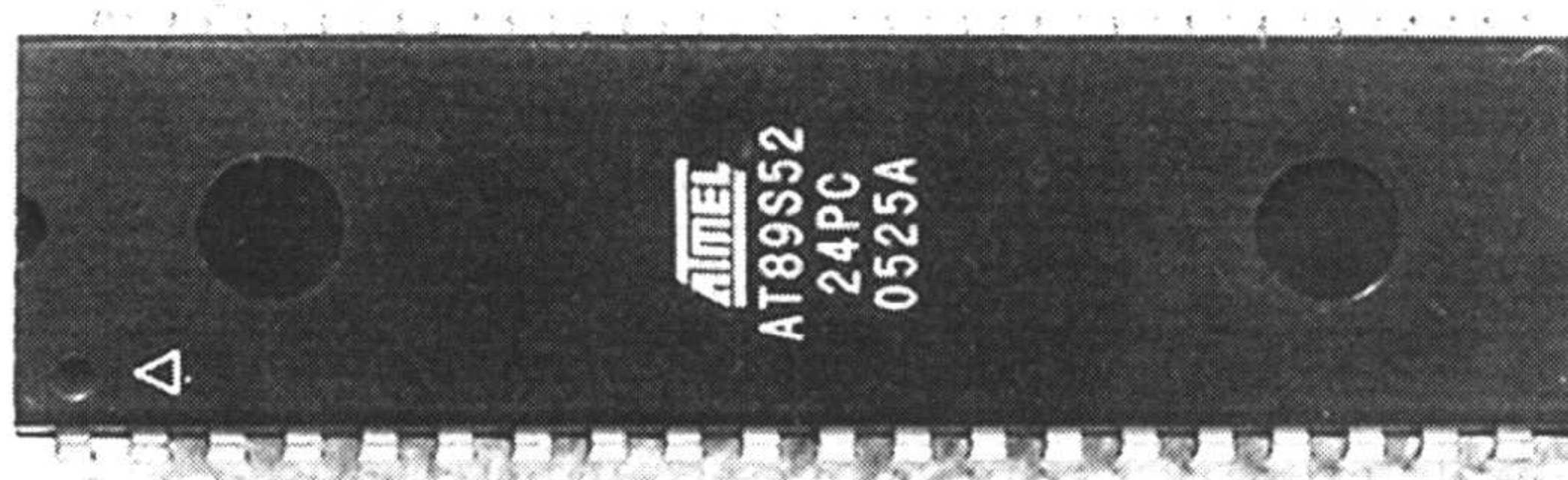


TQFP (Thin Plastic Gull Wing Quad Flat Pack)——塑封超薄封装形式方形贴片式封装,芯片厚度约为 1.00 mm,可直接将引脚敷贴在印制板上焊牢。

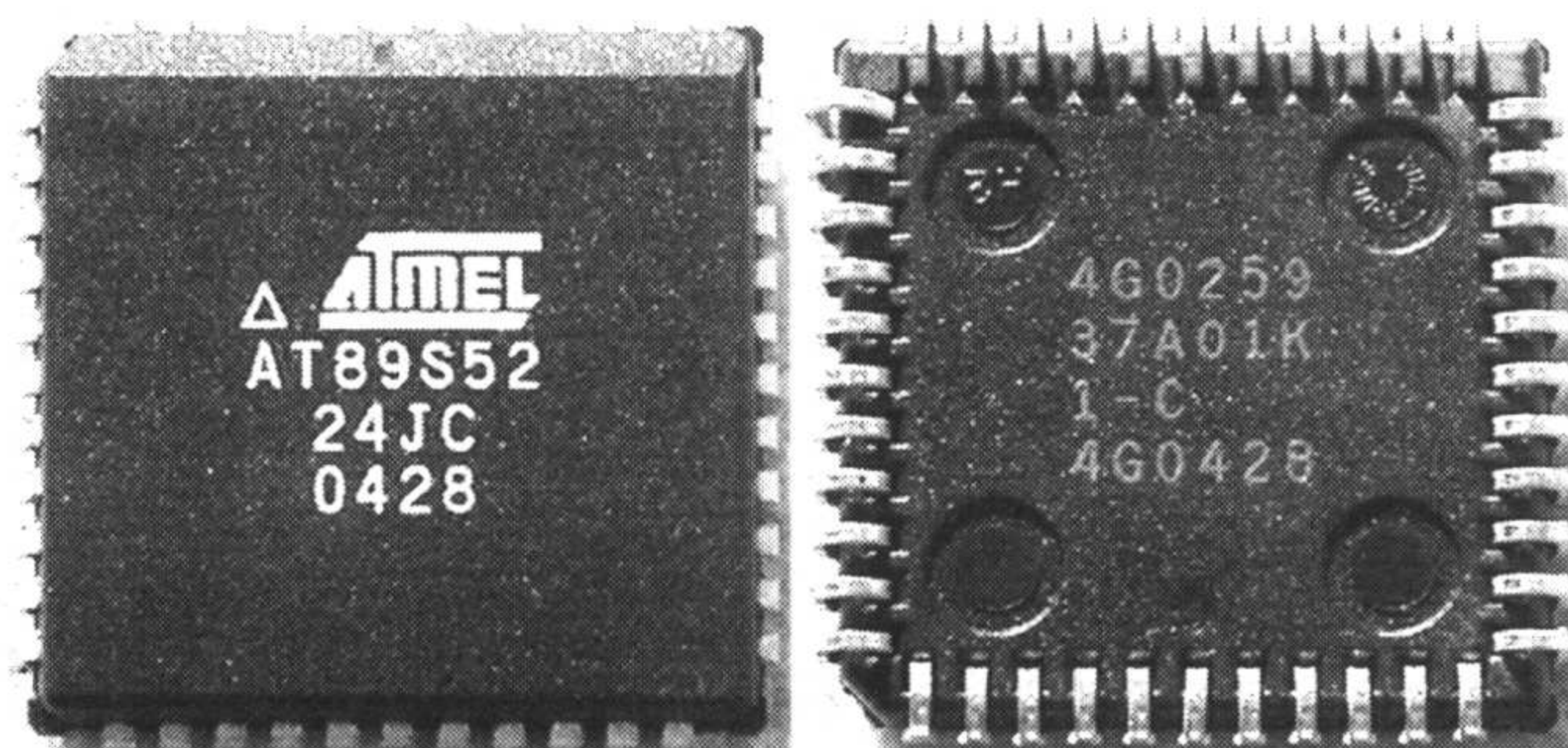
PLCC (Plastic J-Leaded Chip Carrier)——塑封方形引脚插入式封装,可将引脚直接插入到对应的标准插座内。

SOIC (Plastic Gull Wing Small Outline)——双列贴片式封装,可将引脚敷贴在印制板上焊牢。

其中,PDIP、TQFP、PLCC 封装的 AT89S52 如图 1.2.4 所示。



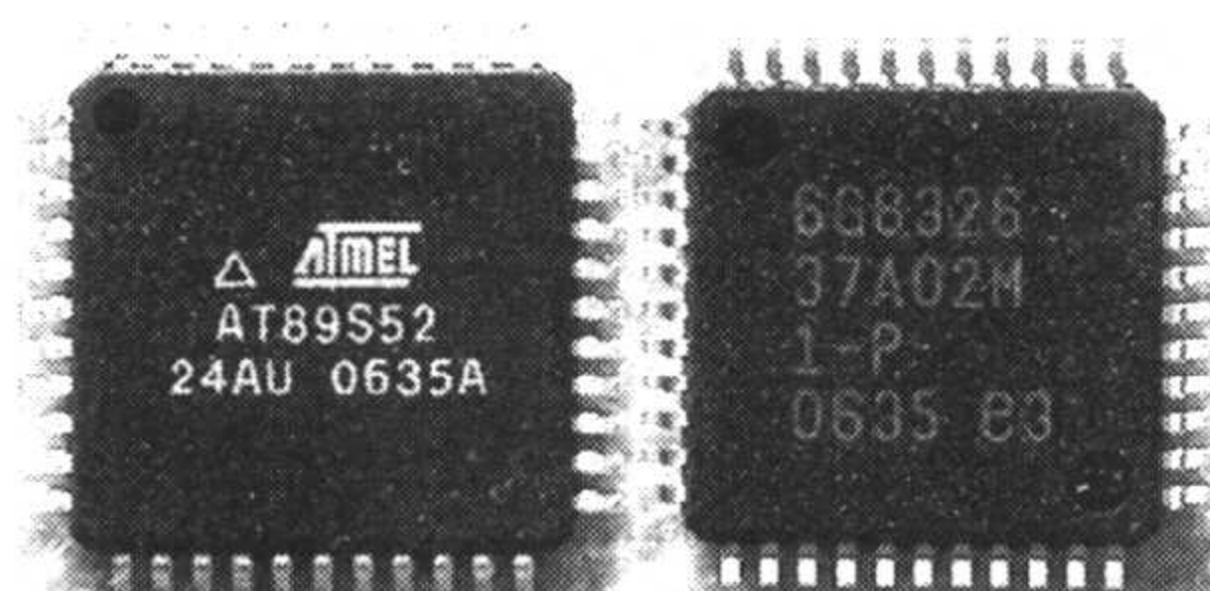
(a) PDIP封装形式的AT89S52



正面

背面

(b) PLCC封装形式的AT89S52



正面

背面

(c) TQFP封装形式的AT89S52

图 1.2.4 AT89S52 封装图

### 1.2.5 部分 ATMEL 单片机的升级替代及推荐产品

由于 IC 制造技术及单片机技术的迅速发展,新的功能更全、性能更好的单片机应运而生,使一些早期的单片机产品由于各种原因已渐渐退出市场,为保证早期开发的产品及设备的正常应用,各公司在推出新的产品时考虑与同类型早期产品的兼容性。ATMEL 公司网站([www.atmel.com](http://www.atmel.com))2006 年提供的不建议在新产品开发中继续使用的单片机型

号及推荐产品如表 1.2.3 所示。

表 1.2.3 ATMEL 公司产品替代及推荐产品表

序号	早期产品	产品描述	替代或推荐产品
1	AT89C51 <sup>①</sup>	4 KB Flash 的 80C31 系列单片机	AT89S51
2	AT89C52 <sup>①</sup>	4 KB Flash 的 80C32 系列单片机	AT89S52
3	AT89LV51 <sup>①</sup>	2.7 V 工作电压, 4 KB Flash 的 8031 系列单片机	AT89LS51
4	AT89LV52 <sup>①</sup>	2.7 V 工作电压, 4 KB Flash 的 8032 系列单片机	AT89LS52
5	AT89LV53 <sup>②</sup>	低电压, 可直接下载 12 KB Flash 单片机	AT89S8253
6	AT89LS8252 <sup>②</sup>	低电压, 可直接下载 8 KB Flash, 2 KB EEPROM 单片机	AT89S8253
7	AT89S53 <sup>②</sup>	在线编程, 12 KB Flash 单片机	AT89S8253
8	AT89S8252 <sup>②</sup>	在线编程, 12 KB Flash, 2 KB EEPROM 单片机	AT89S8253
9	T89C51RB2 <sup>①</sup>	16 KB Flash 高性能单片机	AT89C51RB2
10	T89C51RC2 <sup>①</sup>	32 KB Flash 高性能单片机	AT89C51RC2
11	T89C51RD2 <sup>①</sup>	64 KB Flash 高性能单片机	AT89C51RD2

注: ① 不推荐在新的产品设计中应用, 可用替代产品。

② 新产品设计中建议采用推荐产品。

### 1.3 单片机的应用

为满足各种各样的系统对控制功能的不同需求, 以单片机为控制核心构成的控制系统在规模、结构、功能等方面将会有很大的不同, 根据规模可概括、笼统地划分为基本应用系统和扩展应用系统两类。

#### 1. 基本应用系统

单片机基本应用系统没有扩展的程序存储器 ROM、数据存储器 RAM、扩展的 I/O 接口等扩展部件, 除单片机外仅配置了电源、时钟电路、输入/输出设备和复位电路, 是最小的单片机应用系统, 如图 1.3.1 所示。

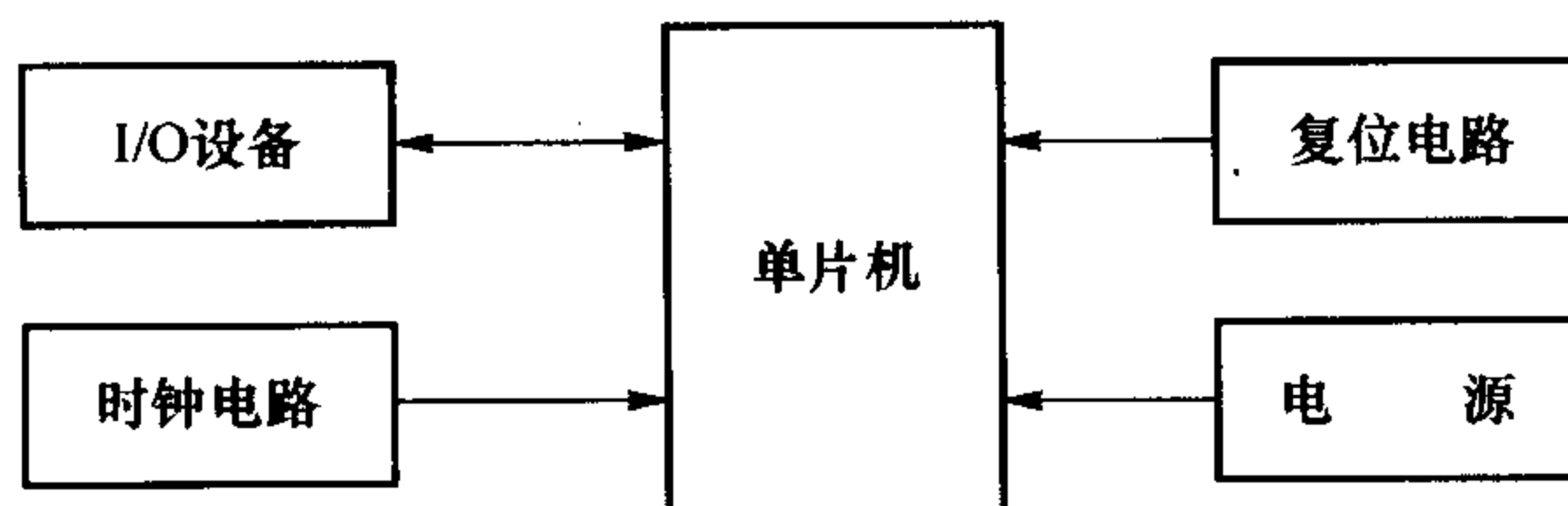


图 1.3.1 最小的单片机应用系统

在许多的智能仪器和设备开发中可以利用单片机最小系统实现。如: 利用最小系统实现的发光二极管循环点亮系统, 各种设备的开关状态检测, 或控制系统执行机构的开关控制(或 PWM 输出)等。



图 1.3.2 为最小系统在锅炉水位控制系统中的应用。其中:L1、L2、L3 分别为正常、超上限、超下限指示灯,用 P0.0~P0.2 控制;正常水位、超上限水位、超下限水位分别由 P1.0~P1.2 检测,为输入信号;超上限报警、超下限报警声音输出由 P1.3、P1.4 控制;加、停水阀门由 P1.5 控制。

单片机通过检测 P1.0~P1.2 状态监测锅炉水位的状态,进而进行指示灯、报警、阀门的控制。正常水位时 L1 点亮,超上限报警时 L2 点亮,并同时声音报警;超下限报警时 L3 点亮,并同时声音报警。

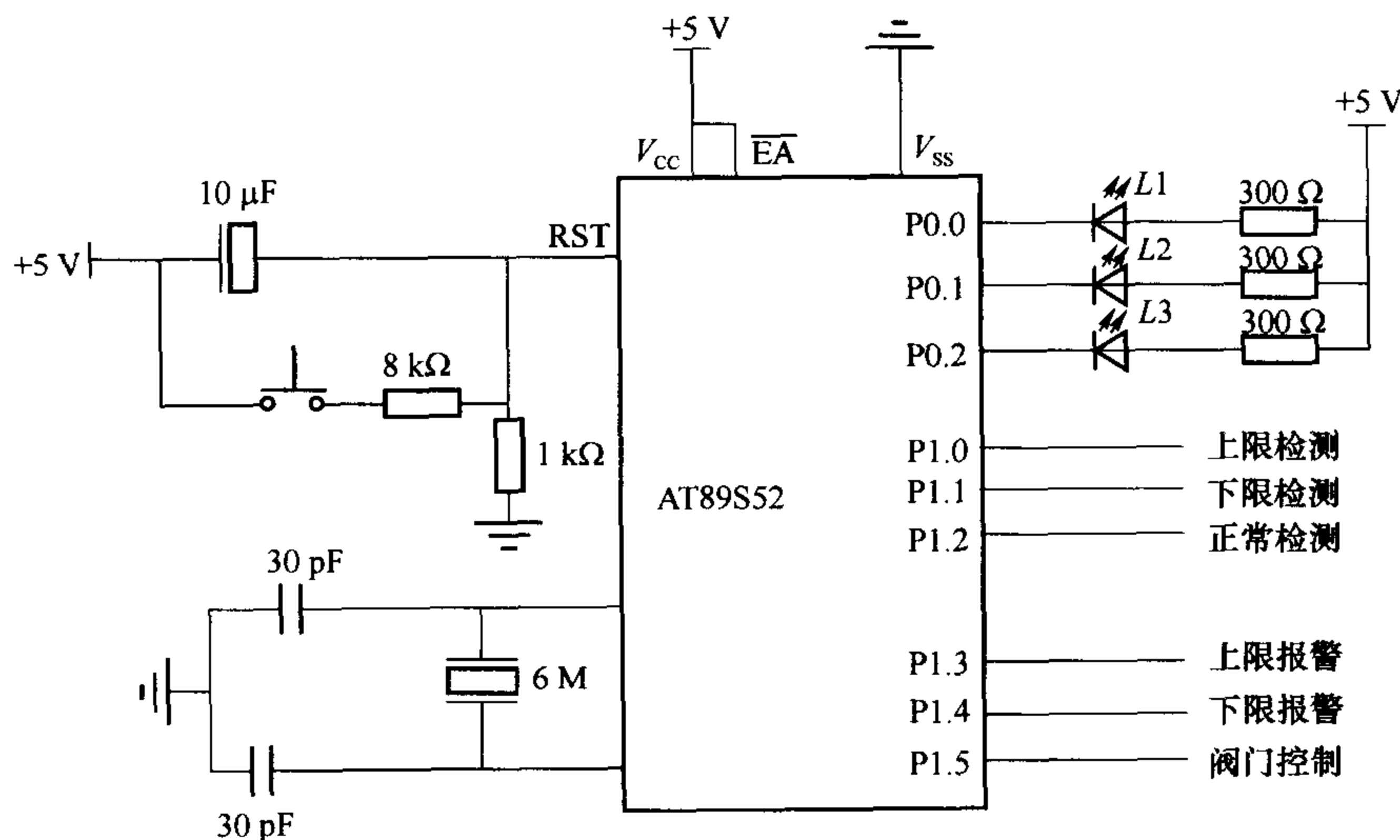


图 1.3.2 最小系统在锅炉水位检测系统中的应用

## 2. 扩展应用系统

当控制系统的功能比较复杂,单片机的内部资源满足不了控制功能的需求时,需要对单片机的资源进行扩展。图 1.3.3 为单片机扩展应用系统的结构图。

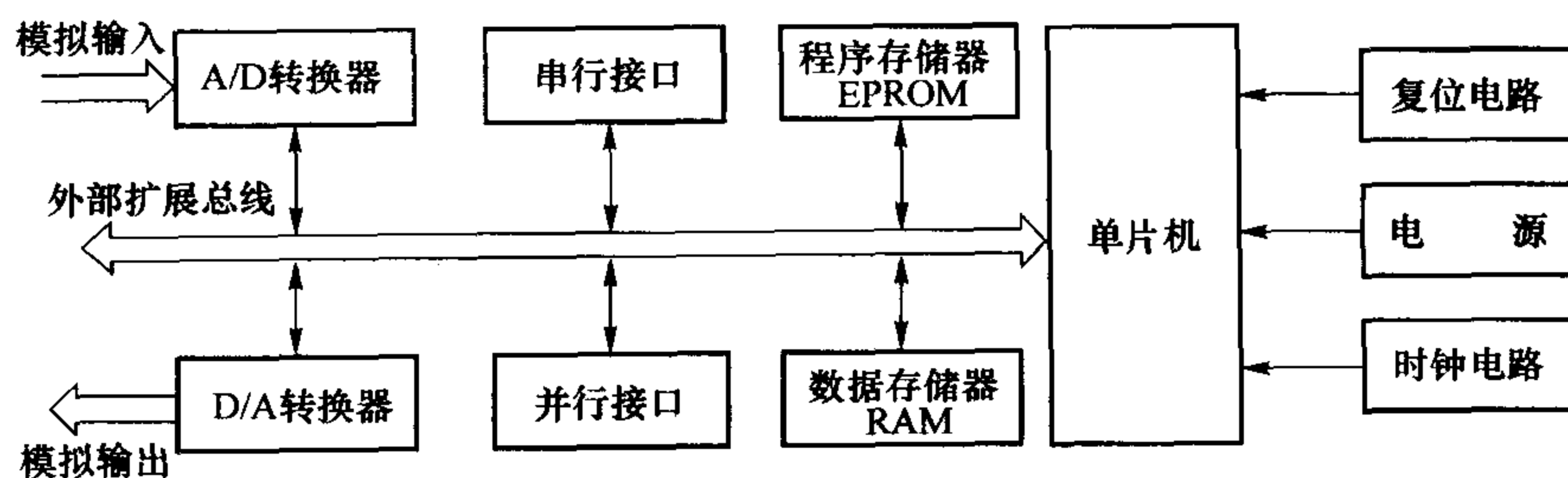


图 1.3.3 单片机扩展应用系统的结构图

从图中可以看出,单片机的扩展以并行口/串行口作为总线,在外部扩展了程序存储器 ROM、数据存储器 RAM、串行口/并行口及 A/D 转换器和 D/A 转换器,以满足各种系统对控制功能的不同需求。

扩展应用系统一般应用在较复杂的系统中。如:利用单片机扩展系统实现的模拟量数据采集,模拟量输出控制,或者与计算机、打印机进行数据的传输和打印等。

图 1.3.4 为利用扩展应用系统实现的锅炉水位检测和控制系統。液位传感器检测液位信号,因液位信号为模拟信号,需通过信号调理电路,然后再经过 A/D 转换器转换成数字信号传输给单片机。单片机根据一定的算法(如 PID)进行计算得到输出控制量(数字量),但因执行机构为模拟器件,所以需经过 D/A 转换再经过信号调理电路进行控制。液位数据可以打印或传输给计算机。键盘用于输入一些控制参数或水位给定值等。液晶(或数码管)显示液位的实时值,输入参数或系统的状态信息等。

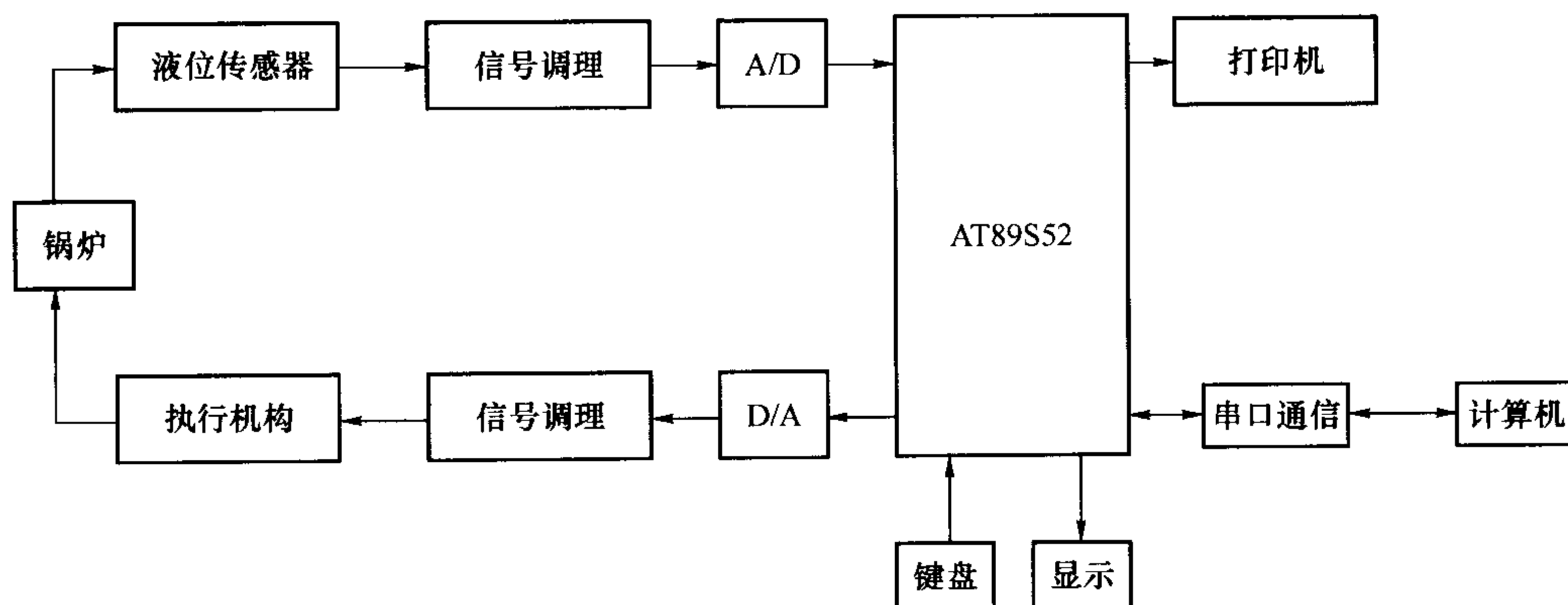


图 1.3.4 扩展应用系统在锅炉水位控制系统中的应用

## 1.4 单片机的发展趋势

AT89 系列单片机从低档型到高档型,基本结构没有发生质的变化,主要是单片机的资源有所增加,伴随着单片机性能的不断f提高和存储器容量的进一步增加。

从第一代单片机到时至今日各大公司推出的各种型号单片机的功能及资源,可以看到单片机正朝着更高性能、更高容量、进一步微型化、多品种、多规格的方向发展,主要体现在如下几个方面。

### 1. CPU 技术的进一步提高

CPU 的运算速度、处理能力进一步提高。为了进一步提高 CPU 的处理能力,双 CPU 结构、增加数据总线的宽度、改进指令的执行队列等技术已经开始应用或正在研发之中。

### 2. 存储器容量的进一步增加和存储器本身技术水平的提高

16 位单片机中 ROM 和 RAM 的容量进一步增大,如飞思卡尔 16 位单片机 MC9S12UF32 具有 32 KB 的 Flash EEPROM、3.5 KB 的 RAM,EPROM、EEPROM、Flash 存储器普遍使用在各种型号的单片机中。特别是 Flash 存储器的使用,使擦除和编程完全是电气实现,大大提高了编程和擦写的速度,并可实现在线编程。

### 3. I/O 口的改进

在单片机发展历程进入第 4 代后,单片机的 I/O 口开始呈现多样化和多功能化的趋势,包括可编程并行口、可编程串行口、串行扩展口、多位的 A/D 转换器、高速输入输出部



件(HSIO)、脉宽调制输出 PWM 等,同时提高并行口驱动能力、增加 I/O 口的逻辑控制功能等多项软硬件技术都已开始应用,这些新技术的应用将会从整体上大幅度提高单片机系统的实时处理能力。

#### 4. 提供特殊的串行口功能

为适应控制系统网络化的需求,一些单片机提供了具有网络功能的特殊串行口,为应用单片机构造控制网络系统提供了便利的条件。

#### 5. 系统的单片化

随着超大规模集成电路制造水平和工艺的不断发展和提高,一些外围电路的功能将会被并入或集成到单片机的芯片内部,包括多位的 A/D 转换器、D/A 转换器、DMA 控制器、频率合成电路、锁相环电路、中断控制器、CRT 控制器等,将一个单片机控制系统集成在一块芯片上。

#### 6. 超小型化

对一些比较简单的设备或系统(例如简单的家电设备、智能化仪器仪表、小单元报警系统等)的控制,并不需要功能特强的单片机来实现,只需要满足控制的需求即可,因此一些功能相对简单、体积更小、功耗小、价格低廉的单片机有着广阔的市场空间,超小型的单片机成为单片机家族的重要组成部分。

#### 7. 微巨机的单片化

美国在 1992 年研发了 i80860 超级单片机,这是一个功能极其强大的单片机,其 CPU 的运算速度达到了 1.2 亿次每秒,可实现 32 位的整数运算和 64 位的浮点数运算,芯片内集成有一个三维图形处理器,i80860 超级单片机配以必要的外设可组成一个超级图形工作站。可以这样认为,随着超大规模集成电路制造水平和工艺的不断发展和提高,今日的高级台式计算机和笔记本电脑会在不久的将来被单片化的计算机取代。

## 第 2 章 AT89S52 单片机的基本结构

ATMEL 公司在 AT89C 系列单片机的基础上,推出了以 MCS-51 核心技术为其内核,采用该公司高性能、低功耗、非易失性存储器技术的 AT89S 系列单片机,包括 AT89S51、AT89S52、AT89S53 和 AT89S8252。与 AT89C 系列相比,AT89S 系列的运算速度有了很大的提高,在功能上新增加了双数据指针、定时监视器(又称看门狗)等,能更好地满足各种不同的应用需要。

本章将重点介绍 AT89S52 单片机的硬件组成结构,AT89S52 的 CPU,AT89S52 的封装及引脚功能、复位电路,AT89S52 振荡器、时钟电路、时序及工作方式。对 AT89S52 的存储器在本章仅作简单的介绍,以使读者对单片机有一个较为系统的认识,更为详细的内容将在第 3 章中讨论。

### 2.1 AT89S52 单片机的主要特性

AT89S52 单片机是 AT89S 系列中的增强型产品,采用了 ATMEL 公司的技术领先的 Flash 存储器,是低功耗、高性能、采用 CMOS 工艺制造的 8 位单片机。AT89S52 单片机的主要特性如下:

- 8 位字长的 CPU;
- 可在线 ISP 编程的 8 KB 片内 Flash 存储器;
- 256 B 的片内数据存储器;
- 可编程的 32 根 I/O 口线(P0~P3);
- 4.0~5.5 V 电压操作范围;
- 3 个可编程 16 位定时/计数器;
- 双数据指针 DPTR0 和 DPTR1;
- 具有 8 个中断源、6 个中断矢量、2 级优先权的中断系统;
- 可在“空闲”和“掉电”两种低功耗方式运行;
- 3 级程序锁定位;
- 全双工的 UART 串行通信口;
- 1 个看门狗定时器 WDT;
- 具有断电标志位 POF;
- 振荡器和时钟电路的全静态工作频率为 0~30 MHz;
- 与 MCS-51 单片机产品完全兼容。

## 2.2 AT89S52 单片机的 CPU

AT89S52 单片机的原理结构如图 2.2.1 所示。AT89S52 单片机的 CPU 8 位字长，主要包括运算器和控制器两部分。

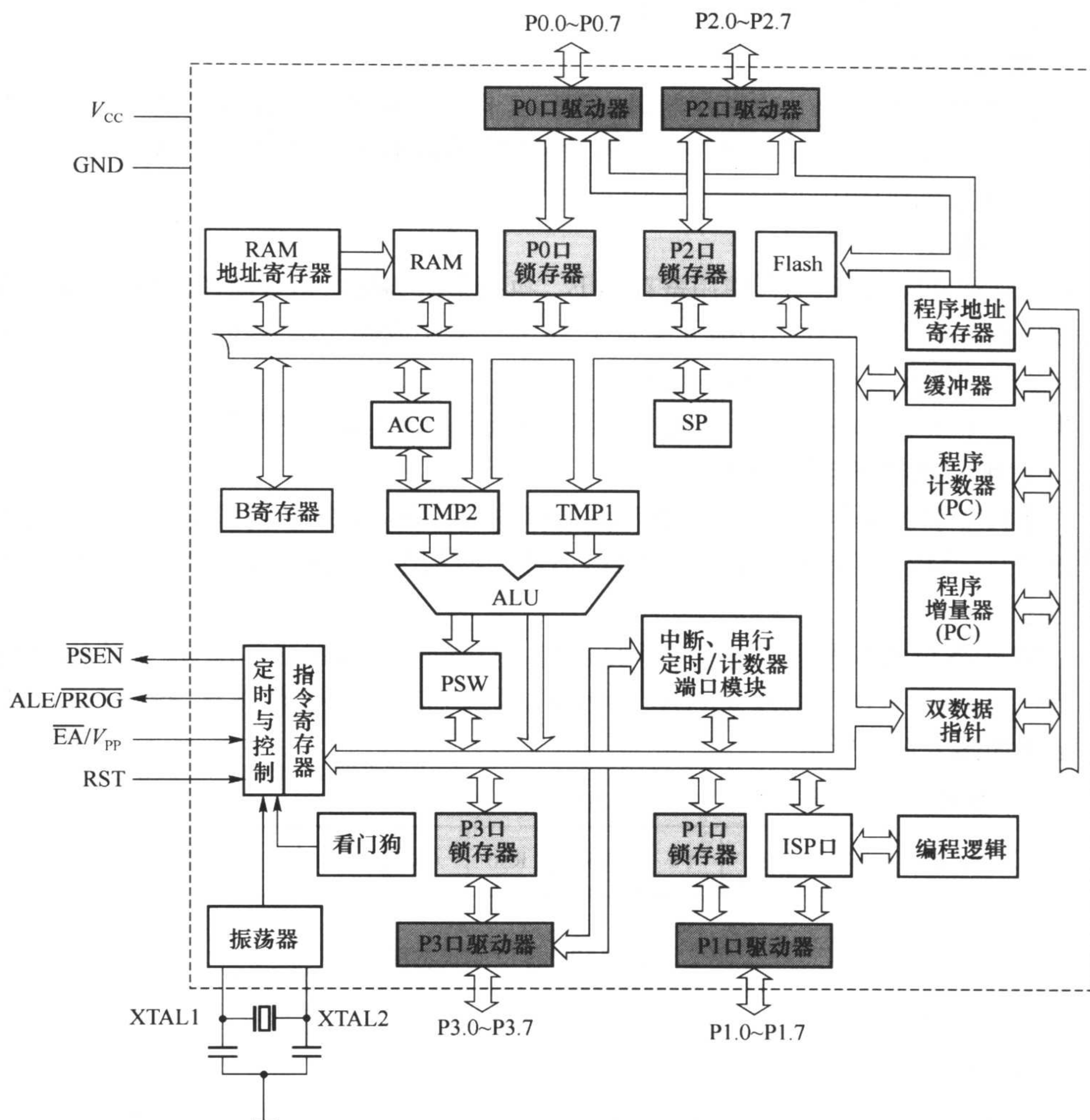


图 2.2.1 AT89S52 单片机原理结构框图

### 2.2.1 运算器

运算器的功能是进行算术逻辑运算、位处理操作和数据的传送，主要包括算术/逻辑运算单元、累加器 ACC、寄存器 B、暂存器 TMP1 和 TMP2、程序状态字寄存器 PSW 等。

#### 1. 算术/逻辑运算单元

算术/逻辑运算单元(ALU)是运算器的核心部件,用来完成基本的算术运算、逻辑运

算和位处理操作。AT89S52 单片机具有极强的“位”处理功能,为用户提供了丰富的指令系统和极高的指令执行速度,除可以进行基本的加、减、乘、除运算外,还可以进行与、非、异或、左移、右移、半字节交换、BCD 码运算、位处理、位检测等。

## 2. 暂存器 TMP1 和 TMP2

从原理结构图中可以看到,运算器中包括的两个暂存器 TMP1 和 TMP2 作为 ALU 的两个输入,暂时存放参加运算的数据。

## 3. 累加器 ACC

累加器 ACC 是一个 8 位寄存器,是 CPU 工作过程中使用频度最高的寄存器。ACC 既是 ALU 运算所需数据的来源之一,同时 CPU 的数据传送大多通过 ACC 实现,因此 ACC 又是数据传送的中间站。

## 4. 寄存器 B

执行乘法和除法指令时,使用寄存器 B。执行乘法或除法指令前,寄存器 B 用来存放乘数或除数,ALU 的另外一个输入来自于 ACC。乘法或除法指令执行完成后,寄存器 B 用来存放乘积的高 8 位或除法的余数。

执行非乘法或除法指令时,寄存器 B 可以作为一般用途的寄存器使用。

## 5. 程序状态字寄存器 PSW

程序状态字寄存器 PSW 是一个 8 位的标志寄存器,用来存放当前指令执行后的有关状态,为以后指令的执行提供状态条件,因此一些指令的执行结果会影响 PSW 的相关状态标志。

PSW 中各位的状态通常在指令执行过程中自动生成,同时 AT89S52 单片机的 PSW 是可编程的,通过程序可以改变 PSW 中各位的状态标志。程序状态字 PSW 各位的状态标志定义如图 2.2.2 所示。

位地址	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H	
PSW	Cy	AC	F0	RS1	RS0	OV	—	P	字节地址 D0H

图 2.2.2 PSW 各位的状态标志

各位定义如下:

Cy:高位进位标志。

若当前执行指令的运算结果产生进位或借位,该标志被置成  $Cy=1$ ,否则  $Cy=0$ 。在执行位操作指令时,Cy 作为位累加器使用,指令中用 C 代替 Cy(详见 4.3.4 小节)。

AC:辅助进位标志位,又称为半字节进位标志位。

在执行加减指令时,如果低半字节向高半字节产生进位或借位,则  $AC=1$ ,否则  $AC=0$ 。

F0:用户标志位。

由用户根据需要进行置位、清 0 或检测。

RS1、RS0:工作寄存器组选择位。

AT89S52 内部数据存储器的容量为 256 B,其中有 4 组工作寄存器,占据了 00H~1FH 的 32 B 存储单元,每组工作寄存器有 8 个工作寄存器,对应符号(R0~R7),每个

工作寄存器既可以用其名称寻址,又可以使用每个工作寄存器的直接字节地址寻址。当使用工作寄存器的名称寻址时,由 PSW 中 RS1 和 RS0 两位给出待寻址工作寄存器所在的组,因此改变 PSW 中 RS1 和 RS0 的内容,便可以选择不同的工作寄存器组(详见 3.2.2 小节)。

OV:溢出标志位。

所谓溢出是指运算结果数值的绝对值超过了允许表示的最大值,该标志位就用来表示有符号数运算时是否产生了溢出。执行运算指令时,如果运算结果超出了目的寄存器 A 所能够表示的符号数的范围( $-128 \sim +127$ ),硬件自动置位溢出标志位,即  $OV=1$ ,否则  $OV=0$ 。该标志的意义在于执行运算指令后,可以根据该标志位的值判断累加器中的结果是否正确。

—:保留位,无定义。

P:奇偶校验标志位。

用来指示累加器中内容的奇偶性,该位始终跟踪指示累加器中 1 的个数,硬件自动置 1 或清 0。若逻辑运算后累加器中 1 的个数为偶数,则  $P=0$ ,否则  $P=1$ 。常用于校验串行通信中数据传送是否正确(详见 9.4.3 小节)。

## 2.2.2 控制器

CPU 中控制器是控制读取指令,识别指令并根据指令的性质协调、控制单片机各组成部件有序工作的重要部件,是 CPU 乃至整个单片机的中枢神经。

控制器由程序计数器 PC、指令寄存器 IR、指令译码器 ID、堆栈指针 SP、双数据指针 DPTR0 和 DPTR1、定时及控制逻辑电路等组成。控制器的主要功能是控制指令的读入、译码和执行,并对指令的执行过程进行定时和逻辑控制。根据不同的指令协调单片机各个单元有序工作。

### 1. 程序计数器 PC

AT89S52 单片机中的程序计数器 PC 是一个 16 位计数器,存放下一条将要执行程序的地址,寻址范围为  $0000H \sim FFFFH$ ,可对 64 KB 的程序存储器空间进行寻址,是控制器中最重要和最基本的寄存器。

系统复位时,PC 的内容为  $0000H$ ,表示程序必须从程序存储器  $0000H$  单元开始执行。

### 2. 指令寄存器 IR

指令寄存器 IR 是专门用来存放指令代码的专用寄存器。从程序存储器读出指令代码后,被送至指令寄存器中暂时存放,等待送至指令译码器中进行译码。

### 3. 指令译码器 ID

指令译码器的功能是根据送来的指令代码的性质,通过定时逻辑和条件转移逻辑电路产生执行此指令所需要的控制信号。

### 4. 堆栈指针 SP

堆栈是一组编有地址的特殊的存储单元,其栈顶的地址由堆栈指针 SP 指示。AT89S52 单片机在片内数据存储器 RAM 中开辟栈区,允许用户通过软件定义片内 RAM 的某一连续区域单元作为堆栈区域。



堆栈指针 SP 是一个 8 位的增量寄存器,所能够指示的深度为 0~255 个存储单元。堆栈操作按照“先进后出”原则进行,数据进栈时 SP 首先自动加 1,然后将欲进栈的数据压入由 SP 指示的堆栈单元;数据出栈时,将 SP 所指示的堆栈存储单元的数据推出栈,然后将 SP 自动减 1。

上电或复位后,堆栈指针 SP 的初始值为 07H,指示栈底为 08H 单元。堆栈指针 SP 的初始值 07H 与工作寄存器组重叠,须通过软件对 SP 重新进行定义,在内部数据存储器 RAM 中开辟一个合适的堆栈区域。

### 5. 双数据指针寄存器 DPTR0 和 DPTR1

在 AT89S52 单片机中,内含 2 个 16 位的数据指针寄存器 DPTR0 和 DPTR1。DPTR0 和 DPTR1 是两个独特的 16 位寄存器,既可以用做 16 位的数据指针使用,也可分开以 8 位的寄存器单独使用(DP0L、DP0H、DP1L、DP1H)。通过软件对特殊功能寄存器 SFR 的辅助寄存器 AUXR1 进行设置,便可以选择 DPTR0 或 DPTR1。AUXR1 是一个不可进行位寻址的特殊功能寄存器,其复位值 =  $\times \times \times \times \times \times \times 0B$ ,地址 = 0A2H。AUXR1 各位定义及格式如下:

字节地址 0A2H	—	—	—	—	—	—	DPS
--------------	---	---	---	---	---	---	-----

DPS:数据指针寄存器选择位。

当 DPS=0 时,选择 DPTR0;当 DPS=1 时,选择 DPTR1。

## 2.3 存储器和 I/O 接口电路

### 1. AT89S52 单片机的存储器

AT89S52 单片机芯片内配置有 8 KB(0000H~1FFFH)的 Flash 程序存储器和 256 B(00H~FFH)的数据存储器 RAM,根据需要可外扩到最大 64 KB 的程序存储器和 64 KB 的数据存储器,因此 AT89S52 的存储器结构可分为 4 部分:片内程序存储器、片外程序存储器、片内数据存储器 and 片外数据存储器。

### 2. I/O 接口电路

CPU 的数据处理速度要远高于外围设备,同时各外围设备的数据处理速度也不尽相同,因此在 CPU 与外围设备之间进行信息交换时要解决处理速度的匹配问题,以有效地提高 CPU 的工作效率;与此同时还要对外围设备进行驱动,I/O 接口电路正是为满足上述要求而设计的。

CPU 和外围设备进行信息交换都要通过接口电路实现。AT89S52 单片机内部集成 4 个可编程的并行 I/O 口(P0~P3),每个接口电路都具有锁存器和驱动器,输入接口电路都具有三态门控制。P0~P3 口同 RAM 统一编址,可以当做特殊功能寄存器 SFR 来寻址。

## 2.4 AT89S52 单片机的封装及引脚功能

AT89S52 单片机有 PDIP、TQFP 和 PLCC 3 种封装形式,本节以 PDIP 封装形式的

AT89S52 单片机为主线,介绍 3 种封装形式的 AT89S52 单片机的引脚及功能。

2.4.1 PDIP 封装的 AT89S52 单片机引脚及功能

PDIP 封装的 AT89S52 单片机引脚排列如图 2.4.1 所示。各引脚的名称、序号及简要功能说明如表 2.4.1 所示。

1. 多功能 I/O 口引脚 P0~P3 口

(1) P0 口 39~32 引脚

8 位并行、双向、开漏输出的 I/O 口,作为输出时可驱动 8 个 TTL 负载。该口内无上拉电阻,由两个 MOS 管串接,既可断开漏极输出又可处于高阻状态,因此称为双向、漏极开路 I/O 口。

对片外程序存储器和数据存储器进行访问时,该口作为低 8 位地址线和数据总线复用。

对片内 Flash 存储器编程时,该口接收指令的字节代码,而对程序进行校验时该口输出指令的字节代码,程序校验时需要外接 10 kΩ 的上拉电阻。

该口作为通用 I/O 口使用时,需要外接上拉电阻。作为输入口使用时,需要对每个引脚写入 1 成为高阻抗输入口,这时该口为准双向 I/O 口。

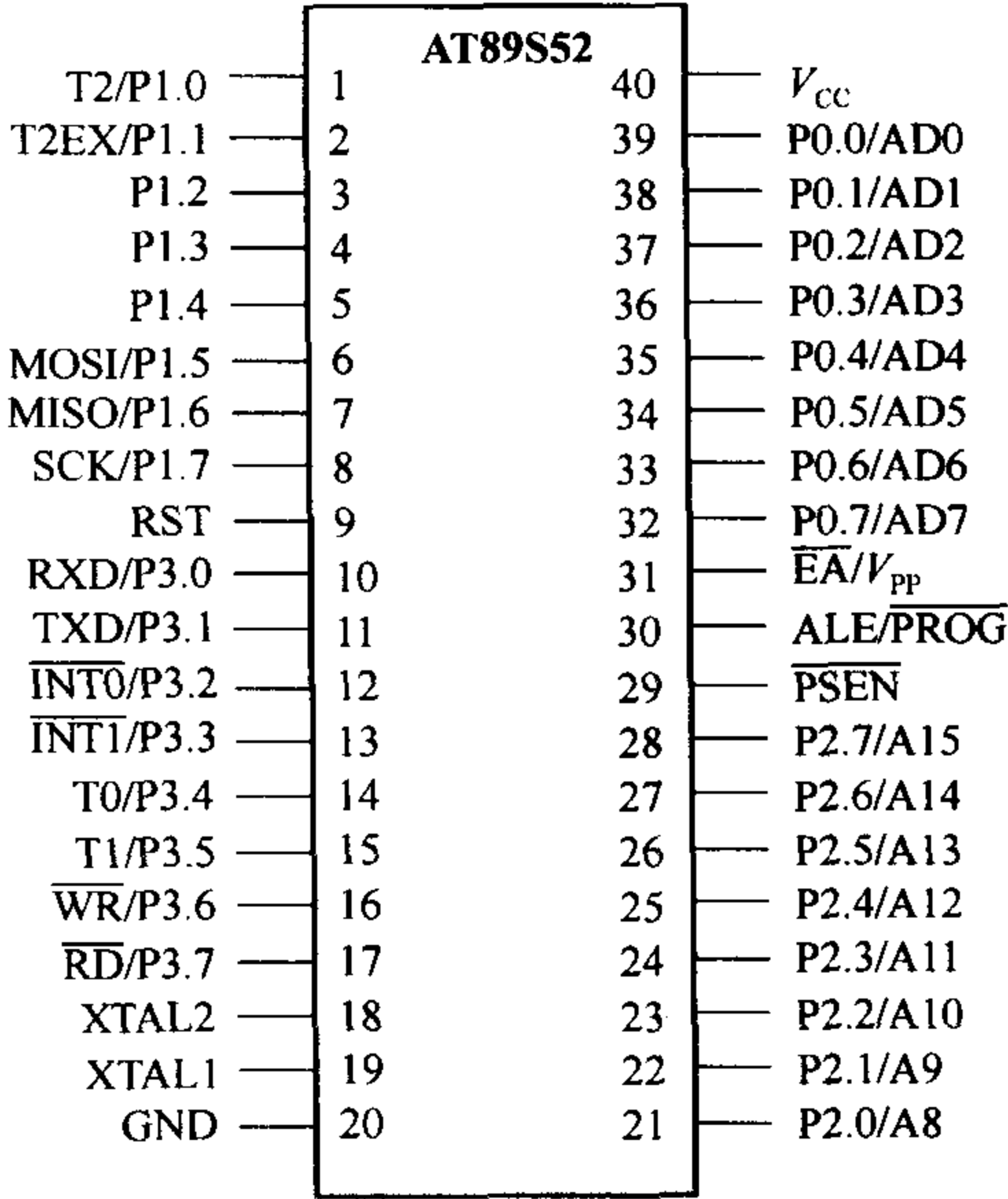


图 2.4.1 PDIP 封装的 AT89S52 单片机引脚排列

表 2.4.1 采用 PDIP 封装形式的 AT89S52 单片机各引脚及功能说明

序号	引脚名称	引脚序号	功能说明
1	P0 口	32~39	8 位并行双向的 I/O 口,访问外部存储器时,可作为低 8 位地址线/数据总线复用
2	P1 口	1~8	用户使用的通用 I/O 口,8 位准双向,编程和校验时作为低 8 位地址线,P1.0 和 P1.1 引脚另有第二功能
3	P2 口	21~28	通用 8 位准双向 I/O 口,访问外部存储器时,可作为高 8 位地址线
4	P3 口	10~17	8 位准双向 I/O 口,还提供了一些第二功能
5	RST	9	复位信号输入端,高电平有效
6	$\overline{\text{EA}}/\text{V}_{\text{PP}}$	31	访问芯片内部和芯片外部程序存储器的选择信号/编程电压
7	$\overline{\text{PSEN}}$	29	外部程序存储器读选通信号,低电平有效
8	$\text{ALE}/\overline{\text{PROG}}$	30	低 8 位地址锁存信号/编程脉冲输入
9	XTAL2 XTAL1	18~19	芯片内振荡器反相放大器的输出端和输入端
10	$\text{V}_{\text{CC}}$	40	电源电压的输入引脚,4.0~5.5 V
11	GND	20	电源地引脚



## (2) P1 口 1~8 引脚

具有内部上拉电阻的 8 位准双向 I/O 口,可驱动 4 个 TTL 负载,当编程和校验程序时定义为低 8 位的地址线。作为输入时需要先将每个引脚置成 1。

引脚 P1.0 同时还是定时/计数器 T2 的外部计数输入,引脚 P1.1 同时还是定时/计数器 T2 捕捉/重装操作的控制信号。

## (3) P2 口 21~28 引脚

具有内部上拉电阻的 8 位准双向 I/O 口,可驱动 4 个 TTL 负载。

访问片外 16 位地址的程序存储器和数据存储器时,该口作为高 8 位地址线。而当只需要 8 位地址时,该口将输出特殊功能寄存器 P2 中的内容。

编程和校验程序时,该口接收高字节地址和一些控制信号。

## (4) P3 口 10~17 引脚

具有内部上拉电阻的 8 位准双向 I/O 口,可驱动 4 个 TTL 负载。

作为普通 I/O 口的输入口使用时,应该先将该口的各引脚写 1。除此之外,P3 口还提供了一些第二功能,如表 2.4.2 所示。

表 2.4.2 AT89S52 单片机 P3 口的第二功能

引 脚	第二功能	说 明
P3.0	RXD	串行数据接收
P3.1	TXD	串行数据发送
P3.2	$\overline{\text{INT0}}$	外部中断 0 请求
P3.3	$\overline{\text{INT1}}$	外部中断 1 请求
P3.4	T0	定时器 0 外部事件计数输入
P3.5	T1	定时器 1 外部事件计数输入
P3.6	$\overline{\text{WR}}$	外部 RAM 写选通
P3.7	$\overline{\text{RD}}$	外部 RAM 读选通

## 2. 复位、控制和选通引脚

共有 4 条控制信号线, RST、 $\overline{\text{EA}}/V_{\text{PP}}$ 、ALE/ $\overline{\text{PROG}}$ 和 $\overline{\text{PSEN}}$ 。

## (1) RST (9 脚)

该引脚为复位信号输入端,高电平有效。在振荡器稳定工作的情况下,将该引脚置成高电平并保持 2 个机器周期以上时,系统复位。

当定时监视器 WDT 溢出时,该引脚置成高电平并持续 96 个振荡周期。

(2)  $\overline{\text{EA}}/V_{\text{PP}}$  (31 脚)

$\overline{\text{EA}}$ 为访问芯片内部和芯片外部程序存储器的选择信号。 $\overline{\text{EA}}$ 为低电平(接地)时,对程序存储器的操作限定在外部程序存储器进行,地址为 0000H~FFFFH。 $\overline{\text{EA}}$ 为高电平(接电源电压  $V_{\text{CC}}$ )时,CPU 首先从芯片内程序存储器(0000H~1FFFH)的 0000H 单元开始读取所存储的指令代码;若芯片外部有扩展的程序存储器,则 CPU 在执行完芯片内部程序存储器中的程序后,自动转向去执行外部程序存储器中的程序。

$V_{\text{PP}}$ 为片内 Flash 存储器的编程电压。对片内 Flash 存储器进行编程时,该引脚接编

程电压  $V_{PP}$  (5 V 或 12 V); 对程序进行校验时, 该引脚接电源电压  $V_{CC}$ 。

### (3) ALE/ $\overline{\text{PROG}}$ (30 脚)

低字节地址锁存允许信号/编程脉冲输入端。ALE 为低字节地址锁存信号, 当系统扩展时 ALE 的下降沿将 P0 口输出的低 8 位地址锁存到外接地址锁存器中, 以实现低字节地址和数据的分时复用。当对 Flash 存储器编程时, 该引脚用作编程脉冲 ( $\overline{\text{PROG}}$ ) 输入端; 在非访问外围器件期间, ALE 连续输出 1/6 振荡频率的脉冲, 可作为外部计数或时钟信号。

### (4) $\overline{\text{PSEN}}$ (29 脚)

$\overline{\text{PSEN}}$  为外部程序存储器读选通信号, 低电平有效。CPU 读取外部程序存储器中的指令代码时, 被读取的指令代码被送至 P0 口; 读写片外数据存储器 RAM 时,  $\overline{\text{PSEN}}$  无效。

## 3. 外部晶振引脚

XTAL1 (19 脚): 芯片内振荡器反相放大器和时钟发生器的输入端。

XTAL2 (18 脚): 芯片内振荡器反相放大器的输出端。

## 2.4.2 PLCC 和 TQFP 封装的 AT89S52 单片机引脚及功能

图 2.4.2 是采用 PLCC 封装形式的引脚排列图, 这是一种方形塑封、引脚插入式封装, 可将引脚直接插入到对应的标准插座内, 引脚数为 44。

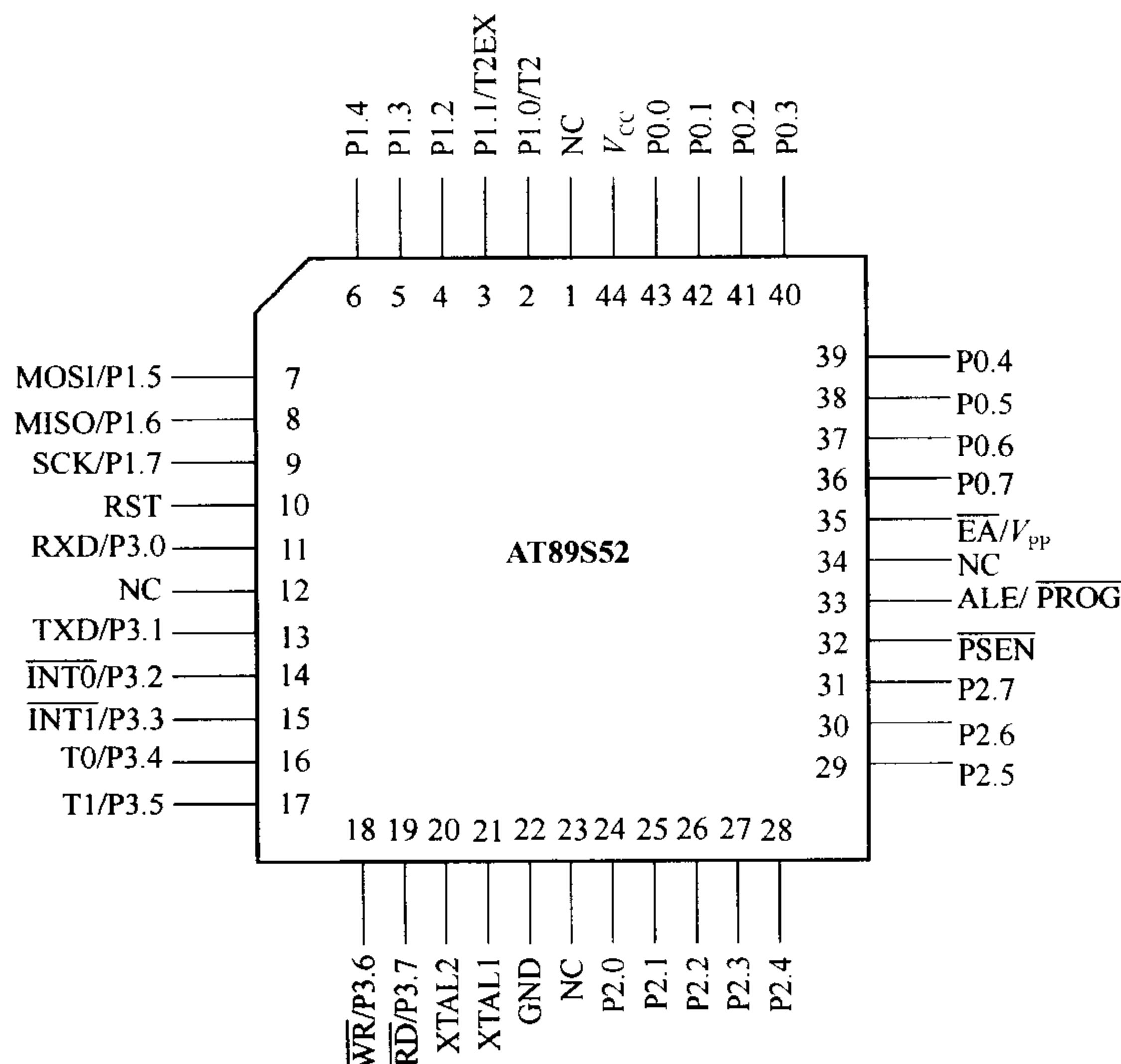


图 2.4.2 PLCC 封装形式的 AT89S52 单片机引脚排列图

PLCC 封装形式各引脚的功能与 PDIP 封装形式各引脚的功能相同, 只是引脚序号与

PDIP 封装形式的引脚序号不同,另外 PLCC 封装形式在方形的四边上各有一个 NC 引脚。

图 2.4.3 是采用 TQFP 封装形式的引脚排列图,这是一种塑封超薄封装形式、方形贴片式封装,芯片厚度约为 1.00 mm,可直接将引脚敷贴在印制板上焊牢。

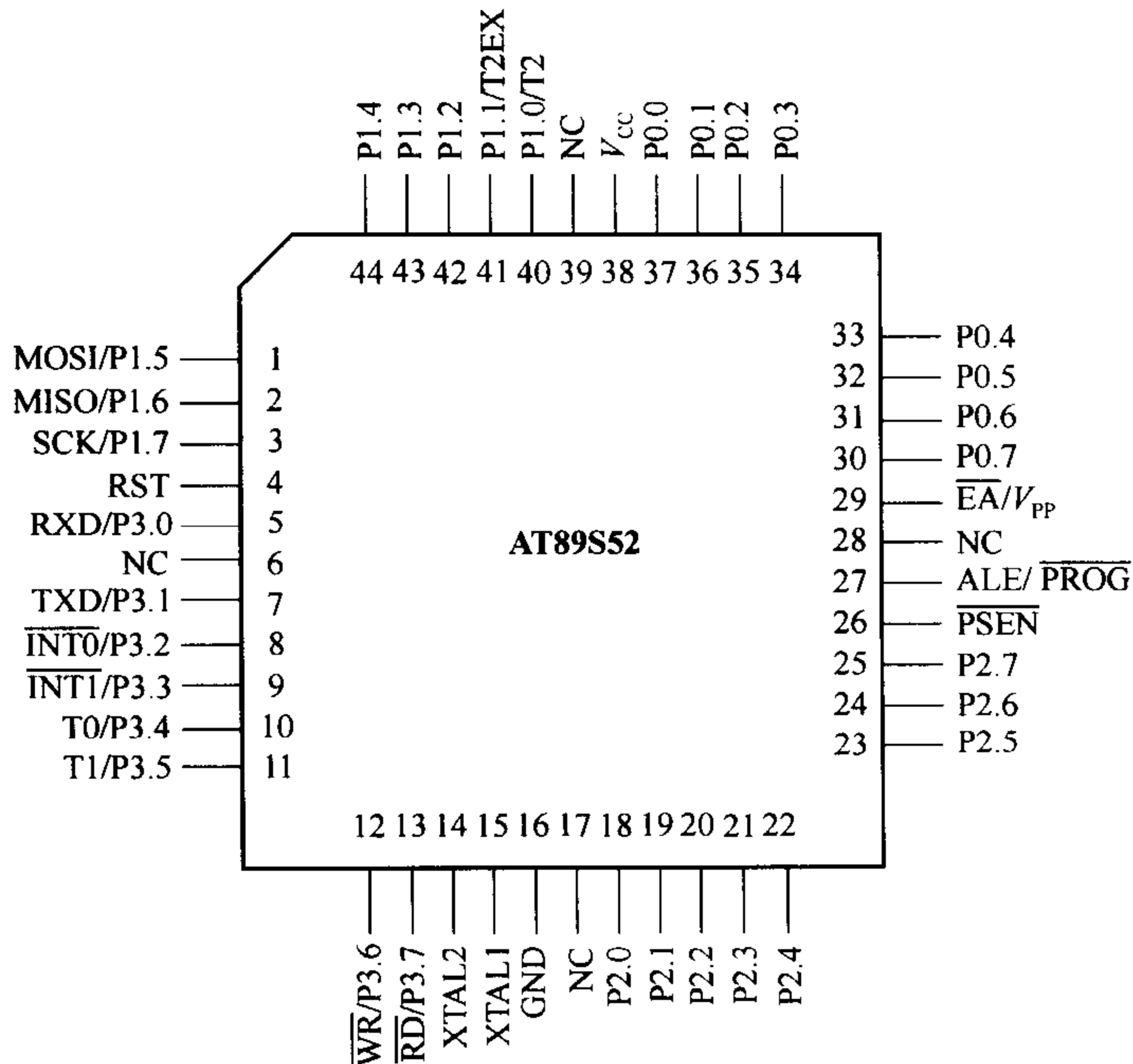


图 2.4.3 TQFP 封装形式的 AT89S52 单片机引脚排列图

TQFP 封装形式的 AT89S52 单片机引脚数量为 44 条,也是在四边上各有一个 NC 引脚。各引脚的功能与 PDIP 和 PLCC 封装形式各引脚的功能相同。

## 2.5 复位操作和复位电路

复位(Reset)操作是使单片机的 CPU 以及系统各部件处于初始状态,并从这个状态开始运行。单片机初始上电后,需要复位操作。单片机在运行过程中可能会受到外界的干扰使程序陷入死循环或“跑飞”,发生这种情况时需要将单片机复位,以重新启动运行。

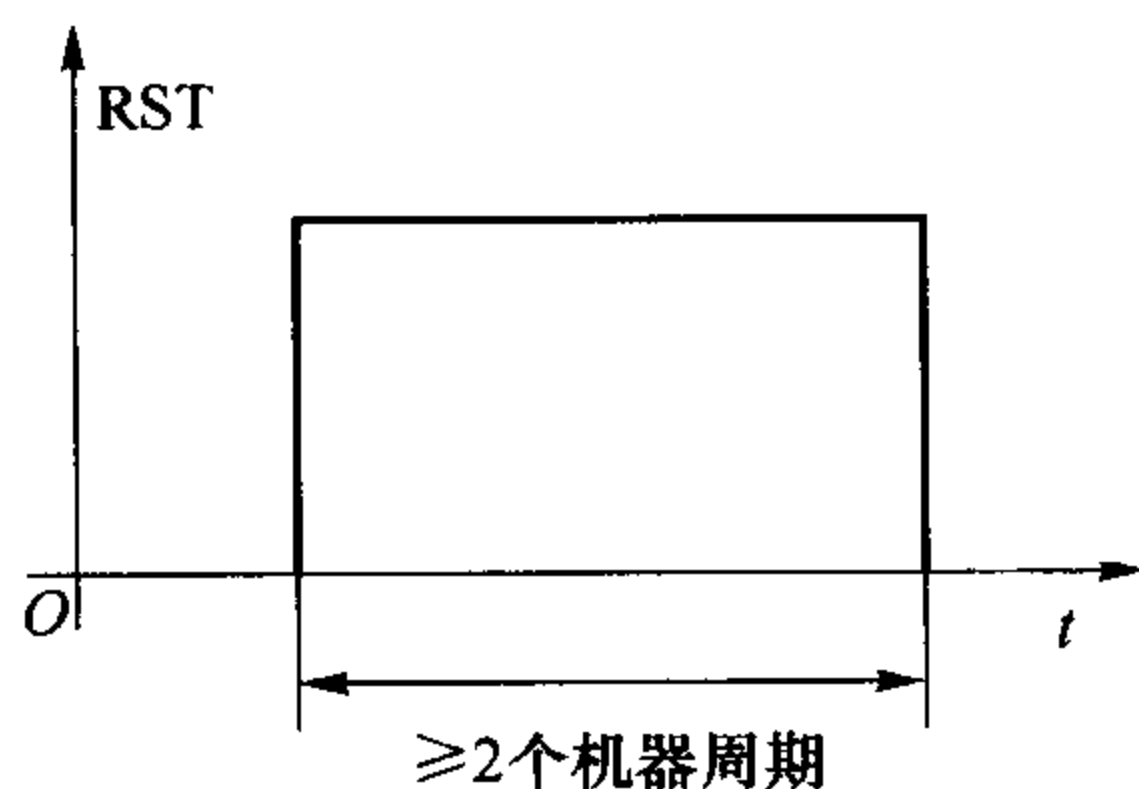


图 2.5.1 复位波形

### 1. 复位操作

RST 引脚是复位信号的输入端口,高电平有效。在时钟振荡器稳定工作的情况下,该引脚若由低电平上升到高电平并持续 2 个机器周期,如图 2.5.1 所示,系统将实现一次复位操作。单片机在 RST 高电平有效后的第二个机器周期开始执行内部复位操作,并在 RST 变为低电平前的每

个机器周期重复执行内部复位操作。

复位操作将使大部分特殊功能寄存器 SFR 置成初始值,如表 2.5.1 所示。

表 2.5.1 特殊功能寄存器 SFR 的复位值

序号	寄存器名称	寄存器符号	复位值
1	程序计数器	PC	0000H
2	P0~P3 口锁存器	P0~P3	FFH
3	堆栈指针	SP	07H
4	数据指针 DPTR0 的低 8 位、高 8 位	DP0L、DP0H	00H
5	数据指针 DPTR1 的低 8 位、高 8 位	DP1L、DP1H	00H
6	电源控制寄存器	PCON	0××× 0000B
7	定时器 0 和 1 控制、模式寄存器	TCON、TMOD	00H
8	定时器 0 低 8 位、高 8 位	TL0、TH0	00H
9	定时器 1 低 8 位、高 8 位	TL1、TH1	00H
10	辅助寄存器	AUXR	×××0 0××0B
11	串行口控制寄存器	SCON	00H
12	辅助寄存器 1	AUXR1	×××× ×××0B
13	中断允许寄存器	IE	0×00 000B
14	中断优先级寄存器	IP	××00 0000B
15	定时器 2 控制寄存器	T2CON	00H
16	定时器 2 模式寄存器	T2MOD	×××× ××00B
17	定时器 2 捕捉/重装寄存器低、高 8 位	RCAP2L、RCAP2H	00H
18	定时器 2 低 8 位和高 8 位	TL2、TH2	00H
19	程序状态字寄存器、累加器、寄存器 B	PSW、ACC、B	00H

复位使特殊功能寄存器 SFR 的内容归于复位值有着重要的意义。

(1) 程序计数器 PC=0000H,复位后从程序存储器的 0000H 单元开始执行程序。

(2) P0~P3 口的复位值=FFH,复位后的各 I/O 口为高电平、双向,可以进行输入或输出操作,单片机运行后锁存器的内容已发生变化,各 I/O 口成为准双向口。

(3) 堆栈指针 SP 的复位值=07H,意味着栈底为 08H 单元,与工作寄存器组占据的存储单元 00H~1FH 发生重叠,需要通过软件对 SP 进行重新定义。

(4) 程序状态字寄存器 PSW 的复位值=0000H,因此其工作寄存器组的选择位 RS1 和 RS0 的值均为 0,表示在复位后选择 0 组工作寄存器。

在表 2.5.1 中没有列出的特殊寄存器 SFR,复位后其值随机或无定义。

## 2. 复位电路

复位操作有手动复位和上电自动复位,图 2.5.2(a)为一种上电自动复位电路,图(b)为具有上电自动复位和手动复位两种操作方式的复位电路。

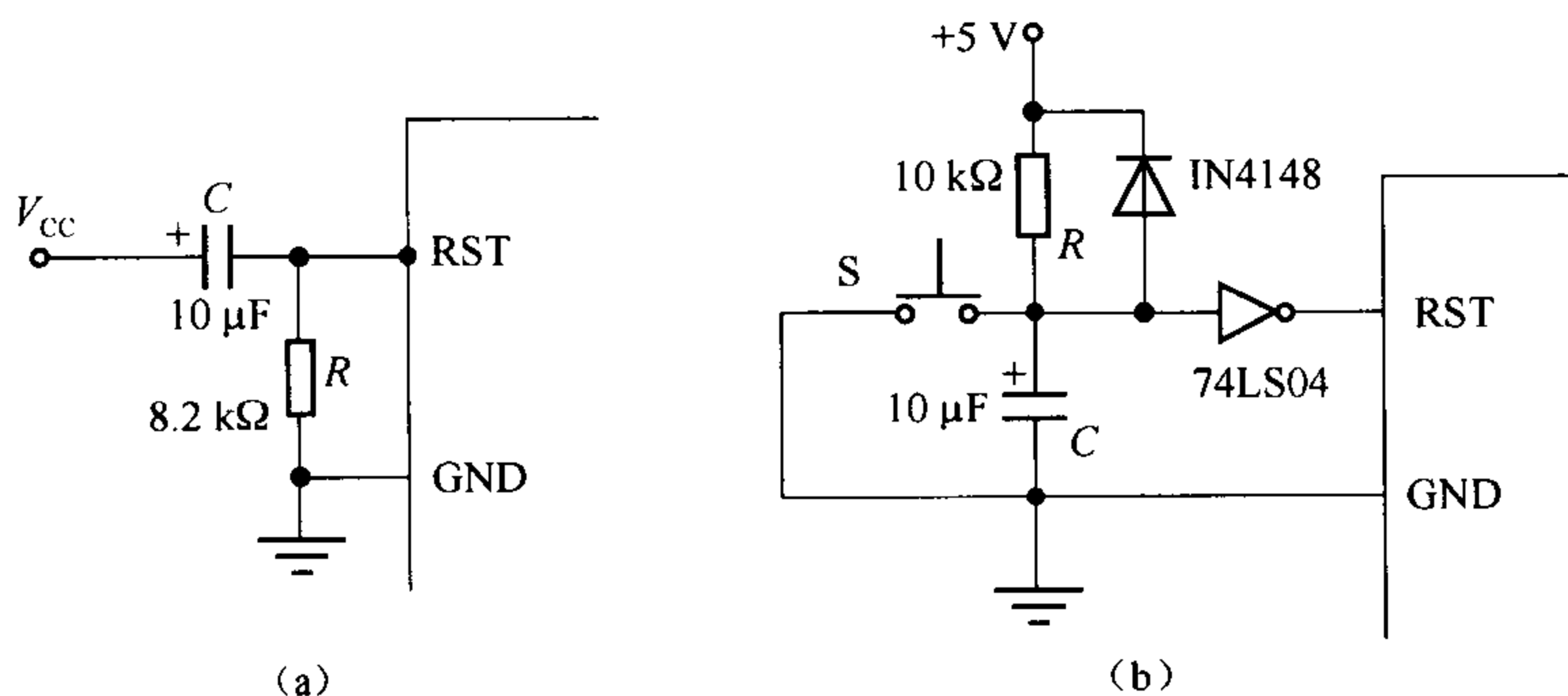


图 2.5.2 两种复位电路

在复位电路上电的瞬间,RC 电路充电,由于电容上电压不能突变,所以 RST 引脚出现高电平。RST 引脚出现的高电平将会随着对电容 C 的充电过程而逐渐回落,为了保证 RST 引脚出现的高电平持续两个机器周期以上的时间,需要合理地选择其电阻和电容的参数值,而电阻和电容参数的取值随着时钟频率的不同而变化。

在单片机应用系统中,除单片机本身需要复位外,外部扩展接口电路等也需要复位,所以系统需要一个同步的复位信号。

为了保证系统可靠工作,CPU 应在系统所有芯片的初始化完成后再对其进行读写。因此硬件电路应保证单片机复位后 CPU 开始工作时,所有的外部扩展接口电路全部复位完毕,即外部扩展接口电路的复位操作完成在前,单片机的复位操作完成在后,也可以采用软件的方式提供这种保证,在主程序的开始部分加入延时,然后再对单片机进行初始化操作。

## 2.6 振荡器、时钟电路及时序

时钟电路用于产生单片机工作所需要的时钟信号,控制着单片机的有序运行节奏,而时序规定了指令执行过程中各控制信号之间的相互关系。在时钟信号的控制作用下,单片机就如同一个复杂的同步时序电路,严格地按照规定的时序进行工作。

### 2.6.1 振荡器

在 AT89S52 芯片内部,有一个振荡器电路和时钟发生器,引脚 XTAL1 和 XTAL2 之间接入晶体振荡器和电容后构成内部时钟方式。也可以使用外部振荡器,由外部振荡器产生的信号直接加载到振荡器的输入端,作为 CPU 的时钟源,称为外部时钟方式。大多数的单片机均采用内部时钟方式,图 2.6.1 为两种方式的电路连接。

采用外部时钟方式时,外部振荡器的输出信号接至 XTAL1,XTAL2 悬空。

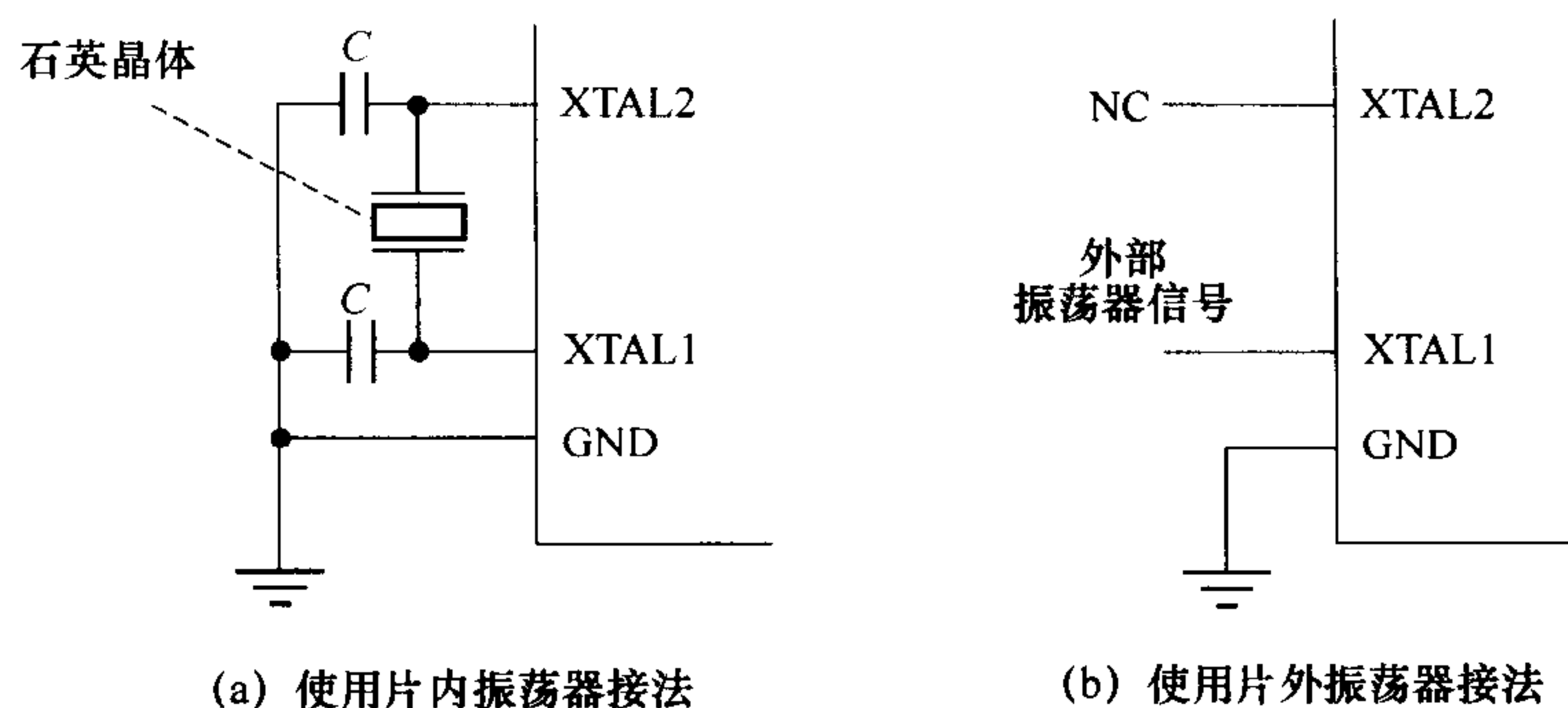


图 2.6.1 AT89S52 振荡器的连接方式

在 AT89S52 单片机内部,引脚 XTAL2 和引脚 XTAL1 连接着一个高增益反相放大器, XTAL1 引脚是反相放大器的输入端, XTAL2 引脚是反相放大器的输出端,振荡器电路的工作原理如图 2.6.2 所示。

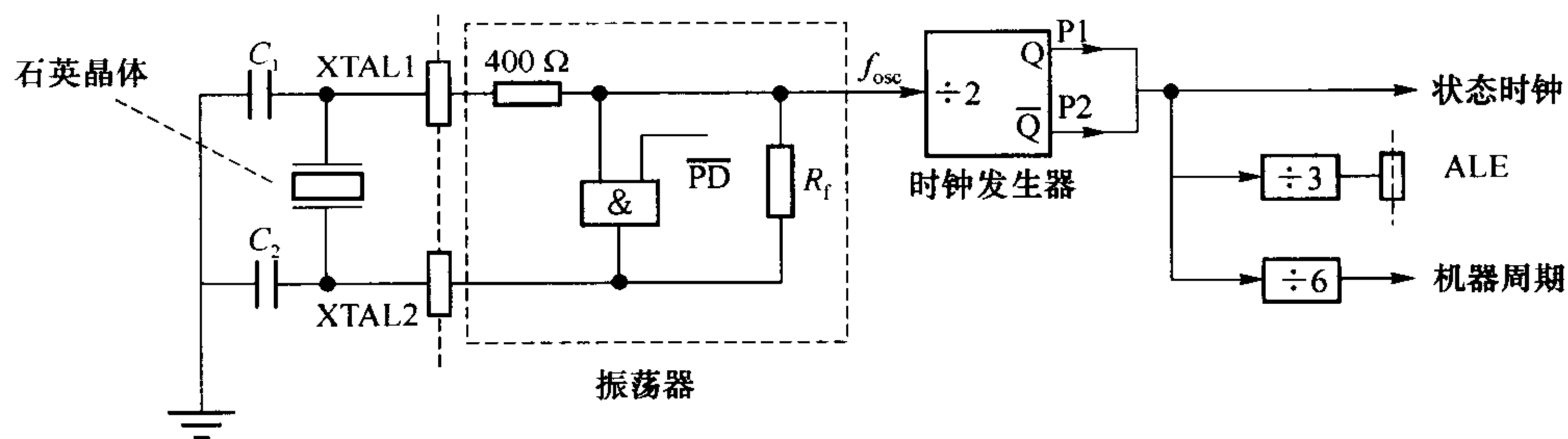


图 2.6.2 AT89S52 振荡器工作原理

芯片内的时钟发生器是一个二分频触发器,振荡器的输出  $f_{osc}$  为其输入,输出为两相的时钟信号(状态时钟信号),频率为振荡器输出信号频率  $f_{osc}$  的  $1/2$ 。状态时钟经三分频后为低字节地址锁存信号 ALE,频率为振荡器输出信号频率  $f_{osc}$  的  $1/6$ ,经六分频后为机器周期信号,频率为  $f_{osc}/12$ 。 $C_1$ 、 $C_2$  一般取  $20\sim 30$  pF 陶瓷电容。

## 2.6.2 AT89S52 的时序

### 1. 时序的定时单位

AT89S52 的时序定时单位有 4 个:节拍、状态、机器周期和指令周期。

#### (1) 节拍 P

节拍 P 又称为振荡周期,分为 P1 节拍和 P2 节拍,节拍的宽度为振荡器输出的振荡信号的周期。P1 节拍通常用来完成算术逻辑操作,P2 节拍通常用来完成内部寄存器之间的数据传送。如果振荡信号是采用内部时钟方式产生的,节拍的宽度就是晶振的振荡周期。如 12 MHz 晶振其振荡周期为  $1/12 \mu s$ 。

#### (2) 状态 S

规定一个状态包含两个节拍,状态的前半个周期对应的节拍称为 P1,后半周期对应的节拍称为 P2。

### (3) 机器周期

如果将一条指令的执行划分为几个基本操作,则完成一个基本操作所需要的时间即为机器周期。规定 6 个状态为一个机器周期,依次表示为 S1~S6。由于一个状态包含两个节拍,因此一个机器周期包含 12 个节拍,表示为: S1P1、S1P2、…、S6P1、S6P2,如图 2.6.3 所示。

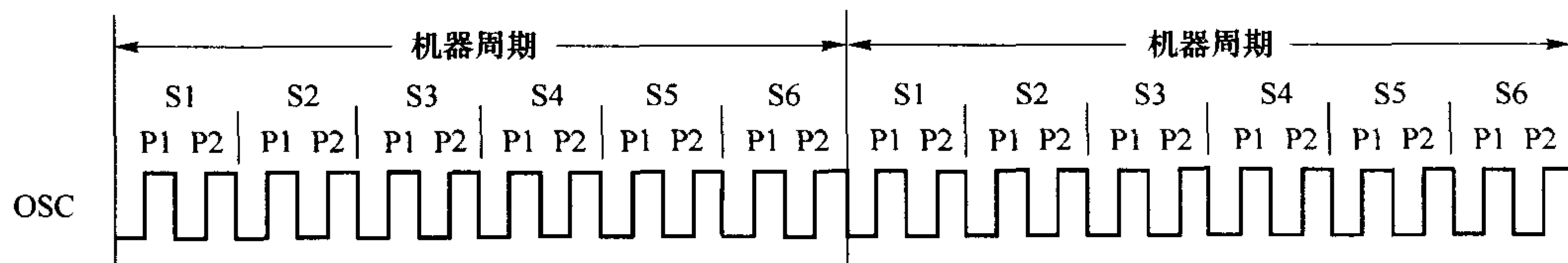


图 2.6.3 AT89S52 的时钟信号和定时单位

从图中可以看到,一个机器周期共有 12 个振荡周期,即机器周期就是振荡器输出信号  $f_{osc}$  12 分频后的信号周期。若晶振频率为 6 MHz,则一个机器周期的时间为  $2 \mu s$ 。

### (4) 指令周期

指令周期定义为执行一条指令所需要的时间。指令周期根据所执行指令的不同,可包含 1~4 个机器周期。晶振频率越低,指令执行时间越长。

根据指令代码的长度,可将指令分为单字节指令、双字节指令、三字节指令和四字节指令。执行这些不同长度的指令所需要的机器周期数也是不同的,有单字节单周期指令、双字节单周期指令、单字节双周期指令、双字节双周期指令、3 字节双周期指令和单字节 4 周期指令(单字节的乘除指令)。

## 2. 单片机的指令执行过程

一条指令的执行过程分为读取指令和执行指令两个阶段。读取指令阶段是根据程序计数器 PC 所指示的地址,从程序存储器中读出该地址的指令代码并送至指令寄存器 IR 中,进入执行指令阶段将指令寄存器 IR 中的指令代码送至译码器译码,产生相应的控制信号以完成指令的执行。

### 3. 单字节单周期指令时序

对于单字节单周期指令,从读取指令代码到完成指令的执行只需要一个机器周期,其时序如图 2.6.4 所示。

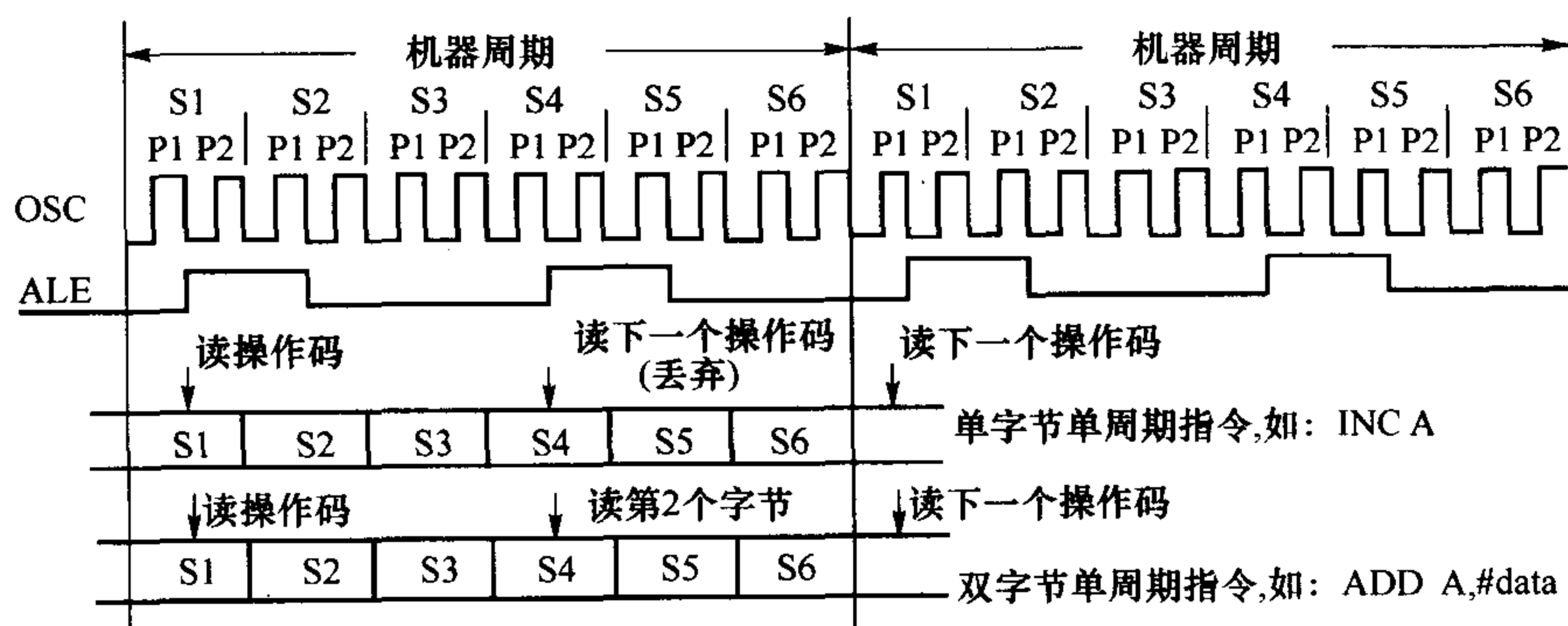


图 2.6.4 典型单/双字节单周期指令时序

在低字节地址锁存允许信号 ALE 第一次有效(S1P2)时,指令代码从程序存储器中读出并被送至指令寄存器 IR 中,从 S1P2 开始执行指令,到第一个机器周期结束时完成指令的执行。由于是单字节指令,因此在 ALE 第二次有效(S4P2)时读取的指令代码将被丢弃,同时程序计数器 PC 的内容也不加 1。

从图 2.6.4 中看到,一个机器周期里 ALE 有效信号出现了两次,这意味着在一个机器周期内可以读取两次程序存储器中所存储的指令代码,提高了处理速度。

#### 4. 双字节单周期指令的时序

图 2.6.4 下部为典型双字节单周期指令时序。ALE 第一次有效时读出指令代码的第 1 个字节, ALE 第二次有效时读出指令代码的第 2 个字节。与单字节单周期指令不同的是,对于双字节单周期指令,两次读取的指令代码均有效,并在一个机器周期的 S6P2 结束时完成指令的执行。

#### 5. 单字节双周期指令的时序

图 2.6.5 为典型的单字节双周期指令时序。

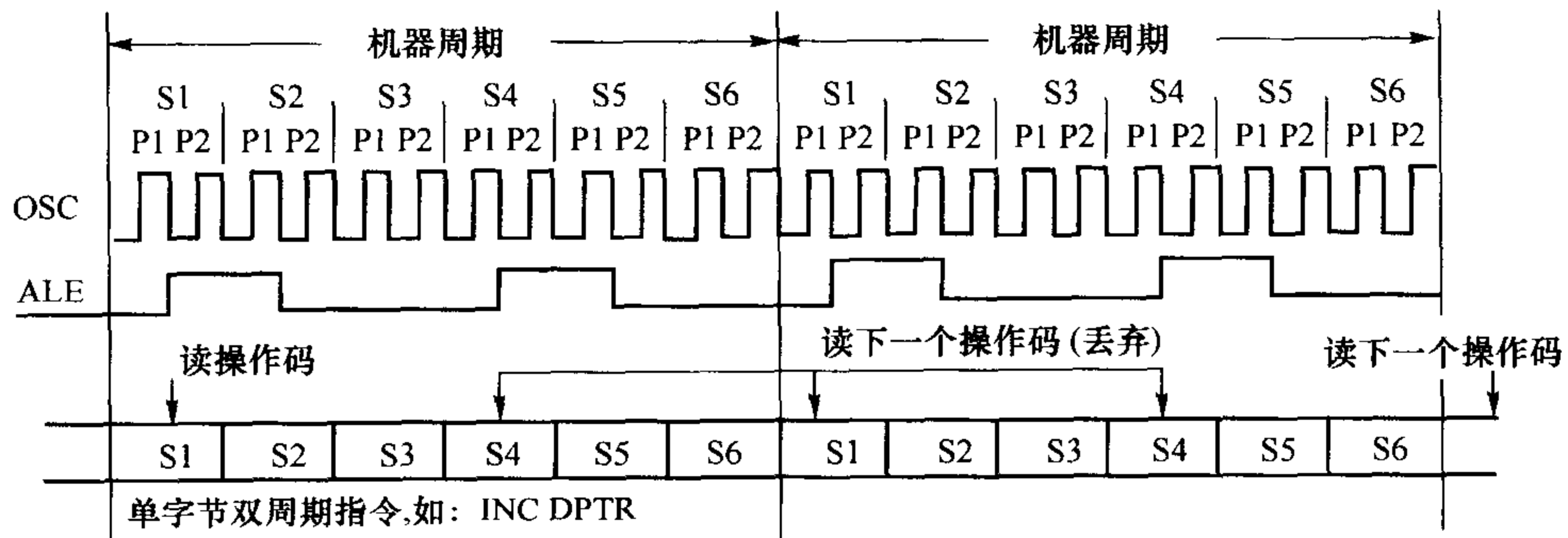


图 2.6.5 典型的单字节双周期指令时序

对于单字节双周期类指令,第一次读取指令代码有效,其余后面的三次读代码均被丢弃,执行该类指令需要 2 个机器周期。

## 2.7 AT89S52 的低功耗工作方式

为了降低单片机运行时的功率消耗,AT89S52 提供了“空闲”和“掉电”两种低功耗工作方式,所以单片机除了正常的程序工作方式外,还可以用低功耗工作方式(又称省电方式)运行。采用 12 MHz 晶体振荡器, $V_{CC}=4.0\sim 5.5\text{ V}$  时,AT89S52 正常工作时的电流最大值为 25 mA,空闲方式的电流最大值为 6.5 mA,掉电方式的电流最大值为 50  $\mu\text{A}$  ( $V_{CC}=5.5\text{ V}$ )。

AT89S52 单片机的两种低功耗工作方式需要通过软件设置才能实现,设置 SFR 中电源控制寄存器 PCON 的 PD 和 IDL 位。电源控制器寄存器 PCON 的格式如图 2.7.1 所示。

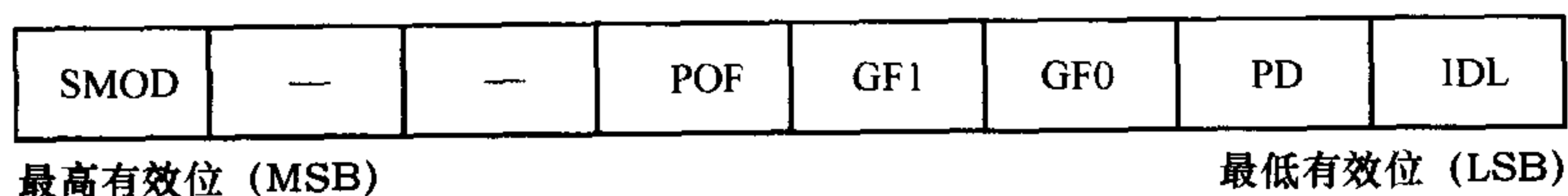


图 2.7.1 电源控制寄存器 PCON 的格式



电源控制寄存器 PCON 是一个不可进行位寻址的寄存器,其复位值=0×××0000B,地址=087H。

各位的功能如下。

SMOD: 波特率倍增位,串行通信时使用。SMOD=1,串行通信工作方式 1、2、3 的波特率加倍;复位时 SMOD=0,原设置的波特率不变。

POF: 断电标志位。

GF1: 通用标志位 1。

GF0: 通用标志位 0。

PD: 掉电方式控制位,PD=1 时进入掉电方式。

IDL: 空闲方式控制位,IDL=1 时进入空闲方式。

电源断电标志位 POF 占据电源控制寄存器 PCON 的第 4 位,当电源上电时将 POF 置 1,POF 也可软件置 1 或者清 0。复位操作对 POF 无影响。

单片机执行完将 IDL 置 1 的指令后,进入空闲工作方式;而将 PD 置成 1 后,单片机进入掉电工作方式。图 2.7.2 为低功耗工作方式的原理图。

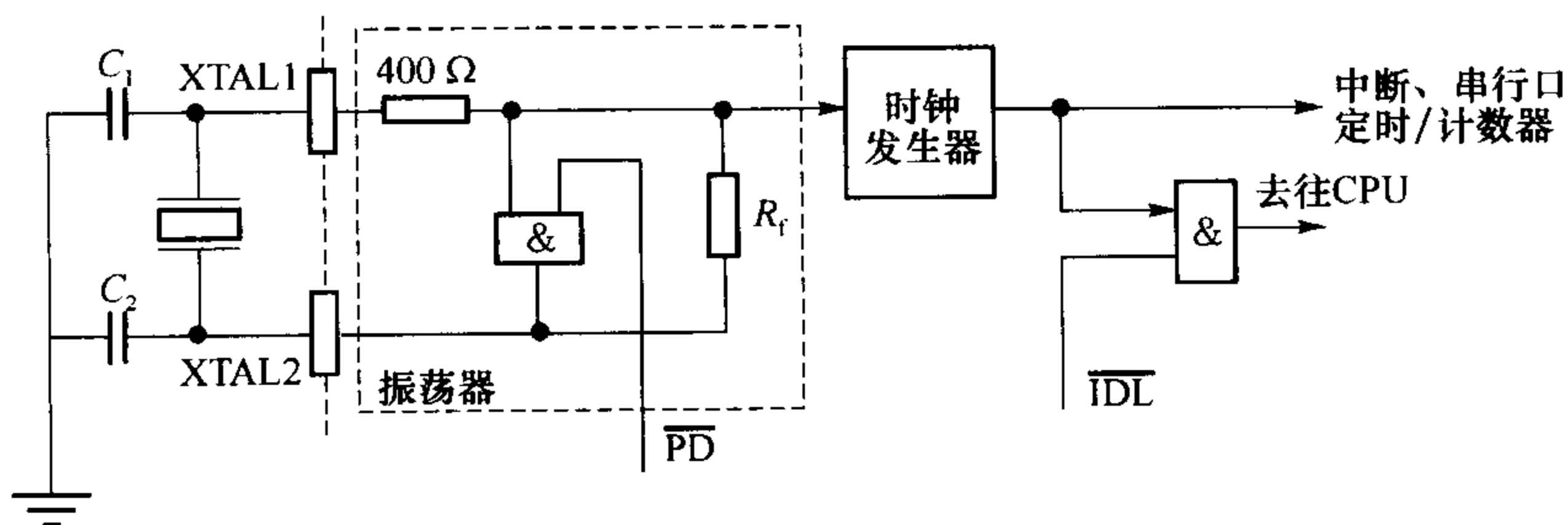


图 2.7.2 低功耗工作方式原理图

### 1. 空闲工作方式

在程序执行过程中,如果不需要 CPU 工作可以让它进入空闲工作方式,其目的是降低单片机的功率消耗。

在空闲工作方式下,IDL=0,IDL=1,单片机的 CPU 停止工作进入休眠状态。此时,振荡器仍然运行,单片机内的所有外设(包括中断系统、定时/计数器、串行口)继续工作。CPU 进入空闲工作方式时,片内 RAM 和所有特殊功能寄存器 SFR 中的内容保持不变,ALE 和 PSEN 的输出为高电平。

AT89S52 单片机退出空闲方式有中断响应方式和硬件复位方式两种。

任何一个可允许的中断申请被响应时,电源控制器寄存器 PCON 的 IDL 位同时会被芯片内的硬件自动清 0,单片机结束空闲工作方式,执行完中断服务程序返回时,从设置进入空闲方式指令的下一条指令处恢复程序的执行,单片机返回到正常的工作方式。

只要 RST 引脚上出现持续 2 个机器周期的复位信号,单片机便可结束空闲工作方式而返回到正常工作方式,并从设置进入空闲方式指令的下一条指令处恢复程序的执行。

需要注意的是:复位操作需要 2 个机器周期的时间才可完成。采用硬件复位方法退出空闲方式时,若 RST 引脚出现复位脉冲,将导致 PCON 的 IDL 清 0,进而退出空闲工作

方式。但退出空闲工作方式所需时间小于 2 个机器周期,即单片机已经退出空闲工作方式并返回到正常工作方式后,复位操作还没有完成。虽然从退出空闲工作方式到复位操作完成期间,复位算法已经开始控制单片机的硬件并禁止对片内 RAM 的访问,但不禁止对端口引脚的访问。为了避免对端口或外部数据存储器等出现意外的写操作,在设置进入空闲工作方式指令后面的几条指令中,应该尽量避免读写端口或外部数据存储器的指令。

## 2. 掉电工作方式

从图 2.7.2 可以看出,当电源控制器寄存器 PCON 的 PD 位置 1 时, $\overline{PD}=0$ ,进入时钟振荡器的信号被封锁,振荡器停止工作,时钟发生器没有时钟信号输出,单片机内所有的功能部件停止工作,但片内 RAM 和 SFR 中的内容保持不变。

AT89S52 单片机退出掉电工作方式也有硬件复位和任何一种有效的外部中断两种方法。

进入掉电工作方式时, $V_{CC}$ 电源电压由正常工作方式下的 +5 V 下降到 +2 V,以达到低功耗运行的目的。退出掉电工作方式前  $V_{CC}$  电源需要恢复到正常的工作电压 +5 V,并维持一个足够长的时间(约 10 ms),以使内部振荡器重新启动并稳定之后才可进行复位操作,以退出掉电工作方式。

采用硬件复位的方法退出掉电工作方式时,将引起所有寄存器的初始化,但不改变芯片内数据存储器 RAM 中的内容。

采用外部中断的方法退出掉电工作方式时,这个外部中断必须使系统恢复到系统全部进入掉电工作方式之前的稳定状态,因此该外部中断启动后约 16 ms 中断服务程序才开始工作。

空闲工作方式和掉电工作方式期间有关的外部引脚的状态如表 2.7.1 所示。

表 2.7.1 空闲和掉电工作方式期间引脚状态

方式	程序存储器	ALE	$\overline{PSEN}$	P0 口	P1 口	P2 口	P3 口
空闲	内部	1	1	数据	数据	数据	数据
空闲	外部	1	1	浮空	数据	地址	数据
掉电	内部	0	0	数据	数据	数据	数据
掉电	外部	0	0	浮空	数据	数据	数据

## 习 题

1. AT89S52 单片机的主要特性有哪些?
2. AT89S52 单片机有哪些主要的逻辑功能部件?
3. AT89S52 单片机的 CPU 中的运算器主要由哪些逻辑部件构成? 这些逻辑部件的主要作用有哪些?
4. 何谓溢出? 溢出后单片机该如何处理?
5. 若选择工作寄存器组 3 区,写出 R0~R7 对应的内部 RAM 单元的字节地址。

6. AT89S52 单片机 CPU 中的控制器主要由哪些逻辑部件构成? 这些逻辑部件的主要作用有哪些?
7. 了解并熟悉 PDIP 形式封装的 AT89S52 单片机各引脚的功能。
8. 何谓复位操作?
9. 复位操作后, AT89S52 单片机有哪些特殊功能寄存器将会被置成初始状态?
10. 复位操作后, AT89S52 单片机程序计数器 PC、堆栈指针和程序状态字寄存器 PSW 的复位值是什么? 这些复位值对单片机有什么意义?
11. 分析复位电路的工作原理。
12. 分析 AT89S52 单片机振荡器的工作原理。
13. AT89S52 单片机的机器周期是怎样组成的?
14. 分析各种字节机器周期指令的时序?
15. 在一个机器周期里信号 ALE 出现了两次, 这意味着什么?
16.  $\overline{\text{PSEN}}$ 、 $\overline{\text{RD}}$  和  $\overline{\text{WR}}$  信号各自的选通作用是什么?
17. 为什么在某些情况下使单片机进入低功耗方式? 如何实现低功耗工作方式?
18. 何谓空闲工作方式? 空闲工作方式的主要特征是什么? 如何退出空闲工作方式?
19. 何谓掉电工作方式? 掉电工作方式的主要特征是什么? 如何退出掉电工作方式?

## 第 3 章 AT89S52 存储器结构

随着超大规模集成电路制造工艺和技术的飞速发展,用于单片机的存储器已经全部实现了半导体集成电路芯片化,在存储速度、存储容量等方面都有了大幅度的提高,出现了采用多种技术、工艺制造的各具优点和特色的存储器。本章将详细介绍 AT89S52 单片机的存储器结构。

### 3.1 存储器概述

#### 1. 各种存储器的特点

表 3.1.1 为各种存储器特点的简要归纳,从中可以了解各种存储器的特点。

表 3.1.1 各种存储器特点归纳

存储器分类	存储器名称	主要特点
半导体随机读写存储器 RAM	双极型 RAM	工作速度快,功率消耗大,集成度低,价格高,仅适合应用于对运算速度要求很高的计算机
	静态 MOS 型 RAM	集成度较高,功率消耗低,价格便宜,运算速度较快,非破坏性读出不需要刷新,断电后所存数据丢失,其制造工艺限制了集成度的提高
	动态 MOS 型 RAM	集成度高,功率消耗更低,价格便宜,运算速度较快,破坏性读出需要刷新,硬件电路复杂
	集成随机存储器 iRAM	将动态 MOS 型 RAM 和动态刷新控制器集成在一块芯片上,集动态型 RAM 和静态型 RAM 的优点于一体
只读存储器 ROM	掩膜存储器 ROM	其所存储的信息必须由厂家在生产过程中固化,适合批量生产,价格低廉
	可编程只读存储器 PROM	可由用户通过专用固化器自行固化,一经固化不能改写,价格较为便宜
	紫外光可擦写只读存储器 EPROM	可多次重复擦写,固化速度快,操作方便,价格便宜,集成度高。必须脱机擦写,擦写时间长,需要专业的固化器和擦除器
	电可擦写只读存储器 EEPROM	不需要脱机擦写,可在线编程,无须外加编程电源和脉冲,无须专用的擦写设备,擦写速度快,操作简单,容量较小,价格较为昂贵
	电可擦写闪速只读存储器 Flash	除具有 EEPROM 的优点外,其擦写速度极快,掉电后数据不会丢失

#### 2. AT89S52 的 Flash 存储器

AT89S52 单片机的程序存储器采用的是 Flash 存储器,Flash 存储器是一种新型的

存储器,是 EPROM 技术和 EEPROM 技术有机结合的产物。如表 3.1.1 所列,紫外光可擦写只读存储器 EPROM 的最大优点是集成度高,价格便宜,电可擦写只读存储器 EEPROM 的最大优点是不需要脱机擦写,可在线编程,Flash 存储器结合了这两种存储器的技术优点,兼有可在线编程,可多次重复擦写,集成度高,价格便宜的优点。

Flash 存储器可在几秒的时间内完成全片的擦除,编程速度超过 EPROM 和 EEPROM,其典型值为  $10\mu\text{s}/\text{B}$ ,比 EPROM 快了 1 个数量级之多,比 EEPROM 快了 3 个数量级之多。价格上 Flash 存储器具有相当的优势,相同容量的 Flash 存储器其价格不到 EEPROM 的  $1/2$ 。Flash 存储器另外一个突出的优点是支持在线编程,允许芯片在不离开电路板或不离开设备的情况下,实现固化和擦除操作,同时具有较强的抗干扰能力,允许电源有 10% 的噪声波动。

Flash 存储器还具有不易挥发性、访问速度快(约 60 ns)的优点,Flash 存储器的不易挥发性使得在掉电后不需要后备电源供电就可长期可靠地保存数据。Flash 存储器的这些优良特性使得其在问世后得到了非常迅速、广泛的应用。

## 3.2 AT89S52 单片机的存储器结构

AT89S52 单片机的存储器结构分为程序存储器和数据存储器,各自又有芯片内和扩展部分,因此 AT89S52 单片机的存储器结构共分 4 部分,即:片内程序存储器、片外程序存储器、片内数据存储器和片外数据存储器。

### 3.2.1 程序存储器

#### 1. 程序存储器的结构

AT89S52 单片机芯片内配置了 8 KB 的可编程 Flash 存储器,地址为  $0000\text{H} \sim 1\text{FFFH}$ ,可外部扩展到 64 KB,程序存储器的结构如图 3.2.1 所示。

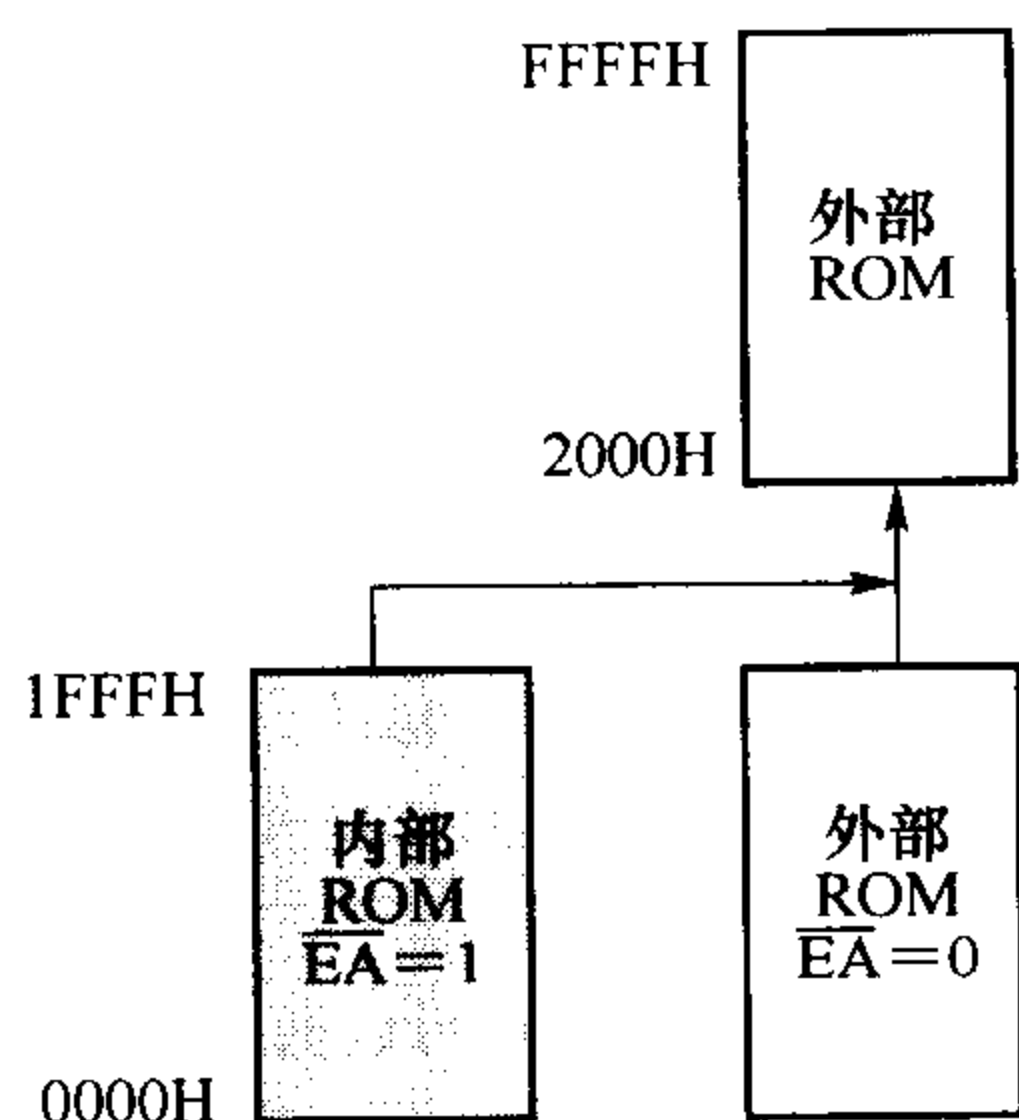


图 3.2.1 AT89S52 程序存储器的结构

从图中可以看到,AT89S52 单片机的程序存储器没有采用分区的方法,64 KB 的地址空间是统一使用的。在读取程序存储器中所储存的程序时,是从芯片内的程序开始读取还是由芯片外的程序开始读取由引脚  $\overline{\text{EA}}$  的信号电平来控制。当引脚  $\overline{\text{EA}}=1$ (接通电源电压  $V_{\text{CC}}$ ) 时,CPU 从芯片内程序存储器的  $0000\text{H}$  单元开始读取指令代码,若外部扩展有程序存储器,且程序计数器 PC 的值超过了  $1\text{FFFH}$ ,则 CPU 会自动到片外程序存储器空间  $2000\text{H} \sim \text{FFFFH}$  读取指令代码。

引脚  $\overline{\text{EA}}=0$ (接地)时,CPU 只对外部程序存储器进行读取操作,由  $\overline{\text{PSEN}}$  信号进行读选通。

程序计数器 PC 是一个 16 位的独立计数器,其中存放着下一条将被读取的指令代码在程序存储器中的地址,程序计数器 PC 中地址数据的变化轨迹决定了控制程序的执行流程。当开机或复位后,PC 中的地址数据为 0000H,因此开机或复位后总是从程序存储器的 0000H 单元开始读取指令代码。

## 2. 程序存储器中的中断矢量区

AT89S52 单片机共有 8 个中断源、6 个中断矢量,当中断源发出中断请求且 CPU 响应中断后便转移到中断服务程序执行。在程序存储器中为中断服务程序保留了一段特殊的区域,即 0003H~0032H 存储单元,专门留给中断服务程序使用,称为中断矢量区,如表 3.2.1 所示。

表 3.2.1 中断服务程序的入口地址

中断源名称	中断标志位	中断矢量地址
外部中断 0( $\overline{\text{INT0}}$ )	IE0	0003H
定时器 0(T0)中断	TF0	000BH
外部中断 1( $\overline{\text{INT1}}$ )	IE1	0013H
定时器 1(T1)中断	TF1	001BH
串行口中断	TI	0023H
	RI	
定时器 2(T2)中断	TF2	002BH
	EXF2	

从表 3.2.1 中可以看到,各中断矢量地址的间隔为 8 B,即为每个中断服务程序留有 8 B 的存储空间。一般情况下这 8 B 容纳不下一段中断服务程序,因此在各中断矢量地址处一般不存放中断服务程序,而是无条件转移指令 AJMP 或 LJMP,执行该指令便能够转去执行中断服务程序,因此在无条件转移指令 AJMP 或 LJMP 中给出的地址才是实际存放中断服务程序的存储单元地址,可以认为中断矢量区实际存放的是中断服务程序的入口地址。

另外当上电或复位后,CPU 总是从程序存储器的 0000H 单元开始读取指令代码,而 0003H~0032H 为中断矢量区,在 0000H~0002H 之间无法存放所有的程序,因此需要在程序存储器的 0000H~0002H 存储单元存储一条跳转指令,使程序的执行跳过中断矢量区而转到程序的真正起始地址。例如,程序存放的起始地址为 0060H,则必须在 0000H~0002H 存储单元存储一条跳转到 0060H 的指令。

## 3.2.2 数据存储器

AT89S52 单片机的数据存储器地址空间分为芯片内部和外部两个部分,这是与程序存储器相同的地方。与程序存储器不同的是使用了不同的指令访问内部数据存储器 and 外部数据存储器,使用 MOV 类指令访问内部数据存储器,使用 MOVX 类指令访问外部数

据存储器。外部数据存储器的最大地址空间为 64 KB,地址范围为 0000H~FFFFH。

AT89S52 内部数据存储器的容量为 256 B,地址范围为 00H~FFH,其内部数据存储器地址空间分布情况如图 3.2.2 所示。

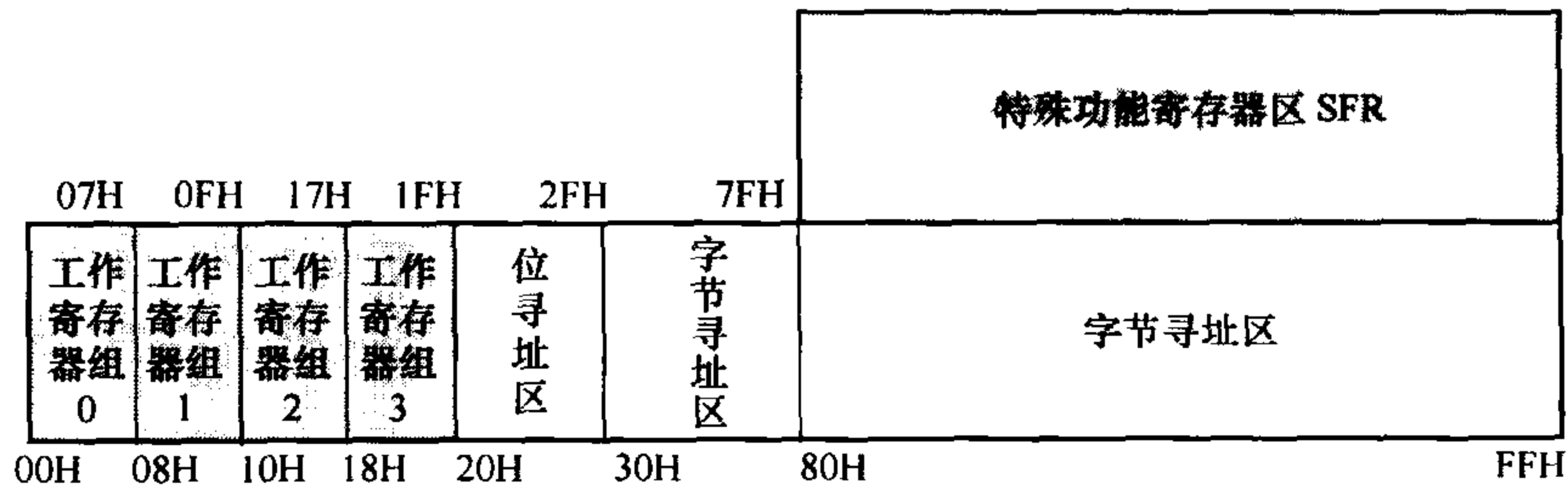


图 3.2.2 AT89S52 单片机内部数据存储器存储空间分布图

从图 3.2.2 中可以看到,整个 256 B 的地址空间分为 4 个区域:工作寄存器区、可按位寻址的 RAM 区、按字节寻址的 RAM 区和特殊功能寄存器区。工作寄存器区内共有 4 组工作寄存器,4 组工作寄存器的地址分别为 00H~07H、08H~0FH、10H~17H、18H~1FH,总共占据了 32 B 的存储单元。每组工作寄存器对应 8 个工作寄存器,其名称为 R0,R1,R2,...,R7。改变程序状态字寄存器 PSW 中 RS1 和 RS0 的内容,便可以选择不同的工作寄存器组,作为 CPU 的当前工作寄存器组。若程序中不需要 4 组工作寄存器,其余的寄存器可做一般 RAM 单元。工作寄存器组一般用于中断服务程序或子程序调用中,为了保护数据,进入中断服务程序后选用与主程序不同的工作寄存器组。工作寄存器组与 RS1、RS0 的对应关系如表 3.2.2 所示。如当前工作寄存器组为工作寄存器组 0,则 R0 对应 00H 单元,R7 对应 07H 单元;若为工作寄存器组 1,则 R0 对应 08H 单元,R7 对应 0FH 单元,依此类推。对 R0~R7 的操作则是对当前工作寄存器组的操作。复位后默认工作寄存器组 0。

表 3.2.2 工作寄存器组与 RS1、RS0 的对应关系

RS1	RS0	寄存器组	片内 RAM 地址
0	0	0	00H~07H
0	1	1	08H~0FH
1	0	2	10H~17H
1	1	3	18H~1FH

内部数据存储器 20H~2FH 共 16 B(128 bit)的存储空间为可按位寻址的 RAM 区域(如表 3.2.3 所示)。可以对这个存储空间按位寻址,即可按位进行读/写操作,这为单片机应用系统的开发带来了极大的便利,用户可以利用这些可按位读/写的 RAM,将应用系统中表示设备运行状态的一些标志存放在这些存储单元中。对于可按位寻址的 16 B RAM 存储空间,未被按位使用的存储单元仍然可以作为一般按字节寻址的 RAM 使用,



充分地利用了有限的数据存储器 RAM 空间。

### 3.2.3 内部 RAM 位寻址区的位地址

单元地址	MSB 位地址						LSB	
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

### 3.2.3 特殊功能寄存器 SFR

AT89S52 芯片内设有 128 B 的特殊功能寄存器区,其特殊功能寄存器有 32 个,与 AT89C51 相比增加了 11 个,与 AT89S51 相比也增加了 6 个。AT89S52 的片内特殊功能寄存器占用 256 B 的高 128 B(80H~FFH)地址,但与 AT89C51 有所不同的是,虽然与芯片内数据存储器 RAM 的高 128 B 地址完全重叠,但两者在物理硬件上是完全独立的,可以用寻址方式来区分这个完全重叠的地址空间。使用直接寻址方式访问这个地址空间时,访问的是特殊功能寄存器 SFR;使用间接寻址方式访问这个地址空间时,访问的是数据存储器。图 3.2.2 明确地标示了地址空间重叠但物理空间完全独立这一结构特点。AT89S52 片内特殊功能寄存器的名称、符号、地址和复位值如表 3.2.4 所示。

表 3.2.4 特殊功能寄存器 SFR 名称、符号、地址和复位值

序号	寄存器名称	寄存器符号	地址	复位值
1	P0 口锁存器	P0*	80H	FFH
2	堆栈指针	SP	81H	07H
3	数据指针 DPTR0 的低 8 位	DP0L	82H	00H
4	数据指针 DPTR0 的高 8 位	DP0H	83H	00H
5	数据指针 DPTR1 的低 8 位	DP1L	84H	00H
6	数据指针 DPTR1 的高 8 位	DP1H	85H	00H
7	电源控制寄存器	PCON	87H	0××× 0000B
8	定时/计数器 0 和 1 控制寄存器	TCON*	88H	00H
9	定时/计数器 0 和 1 模式控制寄存器	TMOD	89H	00H
10	定时/计数器 0 低 8 位	TL0	8AH	00H
11	定时/计数器 1 低 8 位	TL1	8BH	00H
12	定时/计数器 0 高 8 位	TH0	8CH	00H
13	定时/计数器 1 高 8 位	TH1	8DH	00H
14	辅助寄存器	AUXR	8EH	×××0 0××0B
15	P1 口锁存器	P1*	90H	FFH
16	串行口控制寄存器	SCON*	98H	00H
17	串行数据缓冲器	SBUF	99H	×××× ××××B
18	P2 口锁存器	P2*	0A0H	FFH
19	辅助寄存器 1	AUXR1	0A2H	×××× ×××0B
20	WDT 复位寄存器	WDTRST	0A6H	×××× ××××B
21	中断允许控制寄存器	IE*	0A8H	0×00 000B
22	P3 口锁存器	P3*	0B0H	FFH
23	中断优先级控制寄存器	IP*	0B8H	××00 0000B
24	定时器 2 控制寄存器	T2CON*	0C8H	00H
25	定时器 2 模式寄存器	T2MOD	0C9H	×××× ××00B
26	定时器 2 捕捉/重装寄存器低 8 位	RCAP2L	0CAH	00H
27	定时器 2 捕捉/重装寄存器高 8 位	RCAP2H	0CBH	00H
28	定时器 2 低 8 位	TL2	0CCH	00H
29	定时器 2 高 8 位	TH2	0CDH	00H
30	程序状态字寄存器	PSW*	0D0H	00H
31	累加器	ACC*	0E0H	00H
32	寄存器 B	B*	0F0H	00H

表 3.2.4 的顺序是按照寄存器地址的顺序给出的,仔细观察并统计这些寄存器的地址会发现,特殊功能寄存器 SFR 并没有占满 80H~FFH 的存储单元,仅占用了 128 B 中的 32 B 存储单元,而留有一些空闲的存储单元。用户不可以访问这些空闲的存储单元,一旦不慎访问了这些存储单元,将得不到正确的结果,这些存储单元可用于今后新型单片机的研发。

对特殊功能寄存器 SFR 的访问只允许采用直接寻址的指令,否则将会出现错误,这

点在今后编程时应注意。表 3.2.4 中标记 \* 的特殊功能寄存器可以按位寻址(也可按字节寻址),字节地址能被 8 整除,共有 12 个这样的寄存器,除此之外其他的特殊功能寄存器都不能按位寻址。

可按位寻址的特殊功能寄存器的位地址,如表 3.2.5 所示。

表 3.2.5 可按位寻址的特殊功能寄存器的位地址

SFR	最高有效位		位名称/位序号/位地址					最低有效位	字节地址
B	B. 7	B. 6	B. 5	B. 4	B. 3	B. 2	B. 1	B. 0	F0H
	F7	F6	F5	F4	F3	F2	F1	F0	
ACC	ACC. 7	ACC. 6	ACC. 5	ACC. 4	ACC. 3	ACC. 2	ACC. 1	ACC. 0	E0H
	E7	E6	E5	E4	E3	E2	E1	E0	
PSW	Cy	AC	F0	RS1	RS0	OV	—	P	D0H
	PSW. 7	PSW. 6	PSW. 5	PSW. 4	PSW. 3	PSW. 2	PSW. 1	PSW. 0	
	D7	D6	D5	D4	D3	D2	D1	D0	
T2CON	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/ $\overline{T}$	CP/RL2	C8H
	T2CON. 7	T2CON. 6	T2CON. 5	T2CON. 4	T2CON. 3	T2CON. 2	T2CON. 1	T2CON. 0	
	CF	CE	CD	CC	CB	CA	C9	C8	
IP	—	—	PT2	PS	PT1	PX1	PT0	PX0	B8H
	IP. 7	IP. 6	IP. 5	IP. 4	IP. 3	IP. 2	IP. 1	IP. 0	
	BF	BE	BD	BC	BB	BA	B9	B8	
P3	P3. 7	P3. 6	P3. 5	P3. 4	P3. 3	P3. 2	P3. 1	P3. 0	B0H
	B7	B6	B5	B4	B3	B2	B1	B0	
IE	EA	—	ET2	ES	ET1	EX1	ET0	EX0	A8H
	IE. 7	IE. 6	IE. 5	IE. 4	IE. 3	IE. 2	IE. 1	IE. 0	
	AF	AE	AD	AC	AB	AA	A9	A8	
P2	P2. 7	P2. 6	P2. 5	P2. 4	P2. 3	P2. 2	P2. 1	P2. 0	A0H
	A7	A6	A5	A4	A3	A2	A1	A0	
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98H
	SCON. 7	SCON. 6	SCON. 5	SCON. 4	SCON. 3	SCON. 2	SCON. 1	SCON. 0	
	9F	9E	9D	9C	9B	9A	99	98	
P1	P1. 7	P1. 6	P1. 5	P1. 4	P1. 3	P1. 2	P1. 1	P1. 0	90H
	97	96	95	94	93	92	91	90	
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88H
	TCON. 7	TCON. 6	TCON. 5	TCON. 4	TCON. 3	TCON. 2	TCON. 1	TCON. 0	
	8F	8E	8D	8C	8B	8A	89	88	
P0	P0. 7	P0. 6	P0. 5	P0. 4	P0. 3	P0. 2	P0. 1	P0. 0	80H
	87	86	85	84	83	82	81	80	

从表 3.2.5 中可以看到,一些特殊功能寄存器没有位符号,如 P0、P1 等。在位寻址指令中,可以使用“字节地址.位”、“寄存器名称.位”、位地址或者位名称来直接寻址。

### 3.3 外部存储器及其访问

AT89S52 单片机除在芯片内配置了 8 KB 的 Flash 程序存储器和 256 B 的数据存储器外,还可以根据控制系统开发的不同需要,扩展程序存储器和数据存储器。程序存储器和数据存储器最大都可以扩展到 64 KB,访问扩展存储器时 P0 口和 P2 口一起作为对扩展存储器进行寻址的 16 位地址总线,其中 P2 口作为 16 位地址总线的高 8 位地址线,P0 口作为 16 位地址总线的低 8 位地址线。P0 口在作为低 8 位地址总线使用的同时,又作为 8 位的数据总线使用,这时需外接一个地址锁存器,在 ALE 信号的下降沿将低 8 位地址锁存到地址锁存器中,之后释放总线供传送数据或指令代码使用。

#### 3.3.1 外部程序存储器及访问

##### 1. AT89S52 单片机访问外部程序存储器所使用的控制信号

- ALE:低 8 位地址锁存控制信号。
- $\overline{\text{PSEN}}$ :外部程序存储器读取控制信号。
- $\overline{\text{EA}}$ :片内、片外程序存储器访问的控制信号。 $\overline{\text{EA}}=1$  时,访问片内程序存储器; $\overline{\text{EA}}=0$  时,访问片外程序存储器。

##### 2. 访问外部程序存储器的过程

首先通过地址总线给出地址信号,选中程序存储器该地址的存储单元,然后由  $\overline{\text{PSEN}}$  发出读选通信号,在读选通信号的控制作用下,将存储在被选中存储单元中的指令代码读出并送至数据总线,单片机通过对数据总线的访问读取已送至数据总线的指令代码,完成一次对外部程序存储器的访问过程。

##### 3. 扩展外部程序存储器的连接方法

在 AT89S52 单片机的芯片内配置有 8 KB 的 Flash 程序存储器,当不能满足应用系统开发的需要时,可以扩展外部程序存储器,最大可扩展至 64 KB。这样,系统程序存储器的整个存储空间最大可扩展为 64 KB,地址范围为 0000H~FFFFH,其中芯片内配置的 8 KB Flash 程序存储器的地址范围为 0000H~1FFFFH,如图 3.2.1 所示。AT89S52 单片机外部程序存储器扩展方法如图 3.3.1 所示。

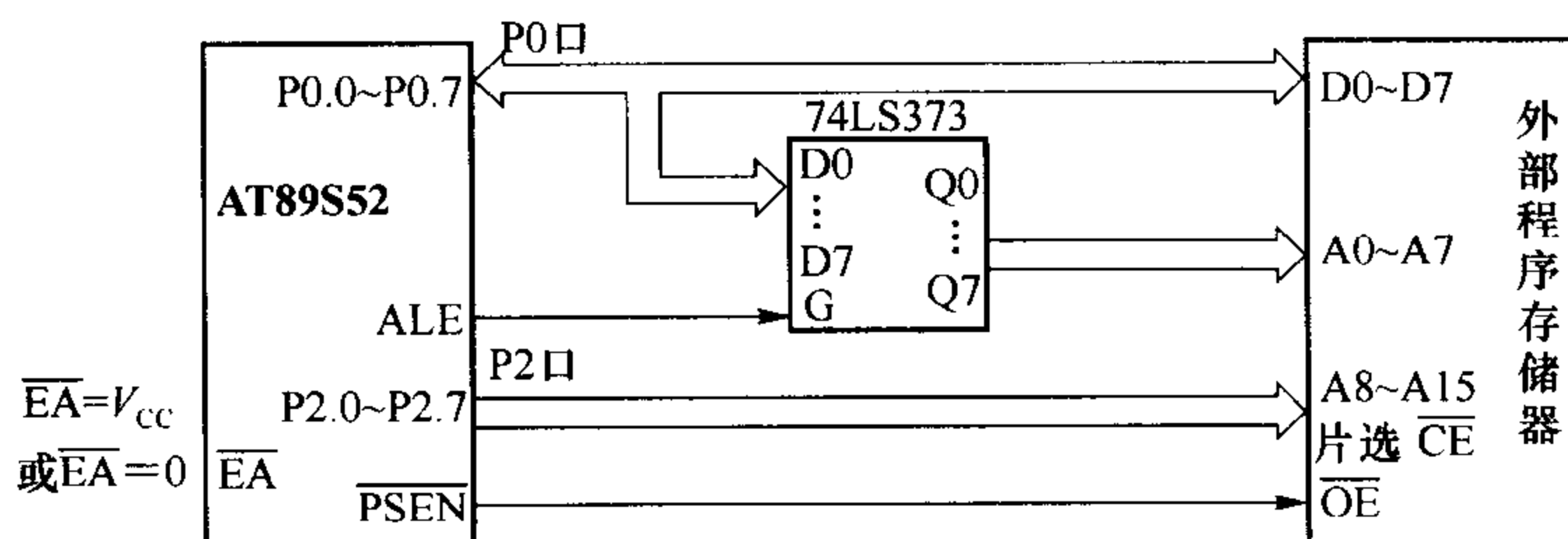


图 3.3.1 外部程序存储器扩展

#### 4. 地址锁存器的作用

当 CPU 访问外部程序存储器时,16 位的程序计数器指针 PC 的低 8 位地址来自于 P0 口,高 8 位地址则来自于 P2 口,P0 口和 P2 口的输出组成了 16 位地址。但由于 P0 口同时又作为由外部程序存储器读取的 8 位指令代码的数据总线,因此为了保证 CPU 访问外部程序存储器期间提供给外部程序存储器的 16 位地址不变,同时由外部程序存储器读取的 8 位指令代码能够通过 P0 口输入到单片机中,必须将由 P0 口输出的低 8 位地址码进行锁存,在 ALE 信号的控制下将其锁存到 74LS373 中,释放 P0 口以接收由外部程序存储器读取的 8 位指令代码。

#### 5. 访问外部程序存储器的时序

CPU 从外部程序存储器读指令时,16 位地址的低 8 位(PCL)由 P0 口输出,高 8 位地址(PCH)由 P2 口输出,而指令的 8 位代码也通过 P0 口输入。CPU 读指令有两种情况:一是不访问外部数据存储器的指令;二是访问外部数据存储器的指令。因此,访问片外 ROM 的操作时序分两种情况:执行非 MOVX 指令的时序;执行 MOVX 指令的时序,如图 3.3.2 所示(MOVX 指令为外部 RAM 访问命令,详见 4.3.1 小节)。

执行非 MOVX 指令时,P2 口专用于输出 PCH 中的内容,P2 口具有输出锁存功能,可直接与外部存储器的地址线相连。P0 口除了输出 PCL 中的内容外,还要输入指令码,所以必须用 ALE 信号锁存 PCL。ALE 信号频率是振荡频率的 1/6,在每个机器周期中,允许地址锁存信号 ALE 两次有效,且在下降沿锁存 PCL。对  $\overline{\text{PSEN}}$  信号而言,也是每个机器周期两次有效,用于选通外部程序存储器,使指令码由 P0 口读入片内。

外部程序存储器的时序分析如下。

(1) 在机器周期的第 3 个振荡周期(S2P1)开始时,单片机将程序计数器 PC 内的 16 位地址信号通过 P0 口和 P2 口送出,P2 口输出 16 位地址信号的高 8 位 A8~A15,并在一个机器周期的第 8 个振荡周期(S4P2)结束前有效。

(2) P0 口输出 16 位地址信号的低 8 位 A0~A7,因为 P0 口还要用来作为数据总线向单片机传送指令代码,所以 P0 口输出的低 8 位地址信号 A0~A7 仅在状态 S2 内保持有效,以便将 P0 口作为数据总线传送指令代码。

(3) 为了保证访问外部程序存储器过程中地址信号的稳定不变,并将 P0 口作为数据总线传送数据,P0 口输出的低 8 位地址信号 A0~A7 需要送至地址锁存器锁存。在地址锁存允许信号 ALE 的下降沿到来时,P0 口输出的低 8 位地址信号 A0~A7 已经稳定,因此可以使用 ALE 的下降沿将低 8 位地址信号 A0~A7 送入地址锁存器锁存。

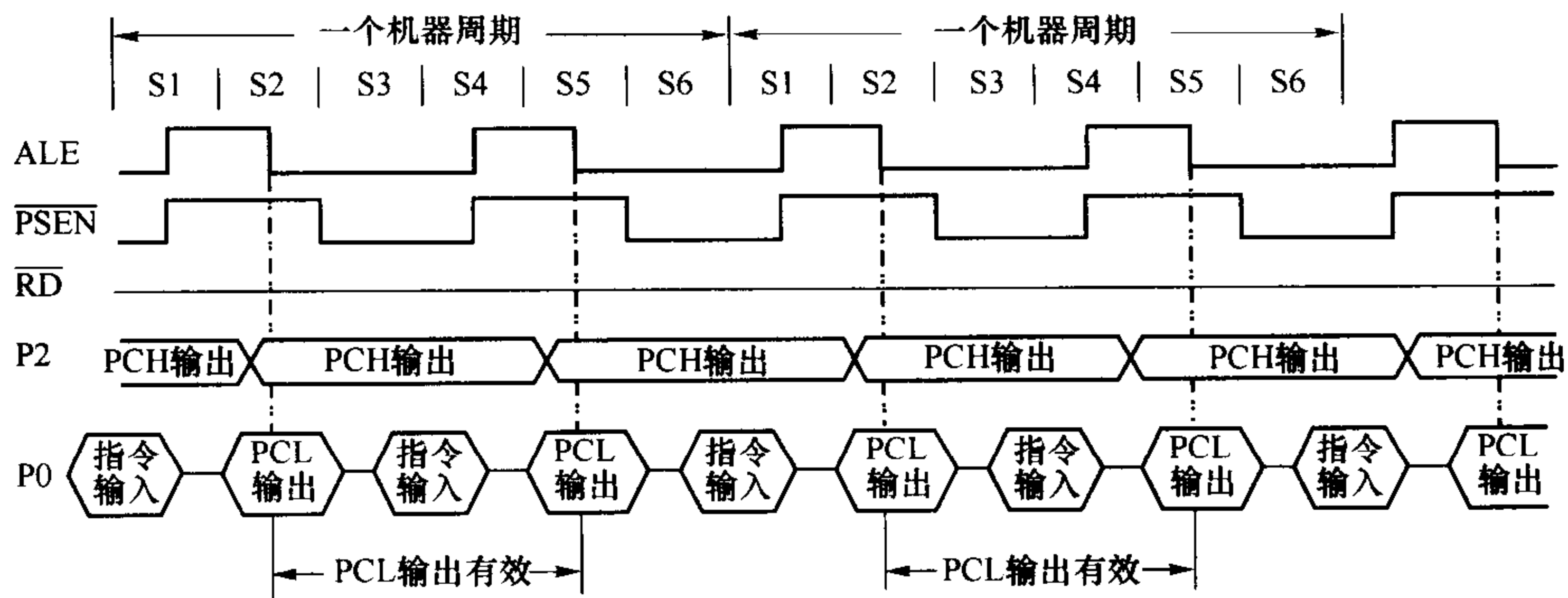
(4) 外部程序存储器读选通信号  $\overline{\text{PSEN}}$  在 S3P1 时已经有效,这时开始选通外部程序存储器,将指定存储单元中的指令代码读至数据总线(P0 口)。

(5) 在  $\overline{\text{PSEN}}$  的上升沿到来前,单片机对 P0 口采样,读入外部程序存储器传送到数据总线(P0 口)上的指令代码。

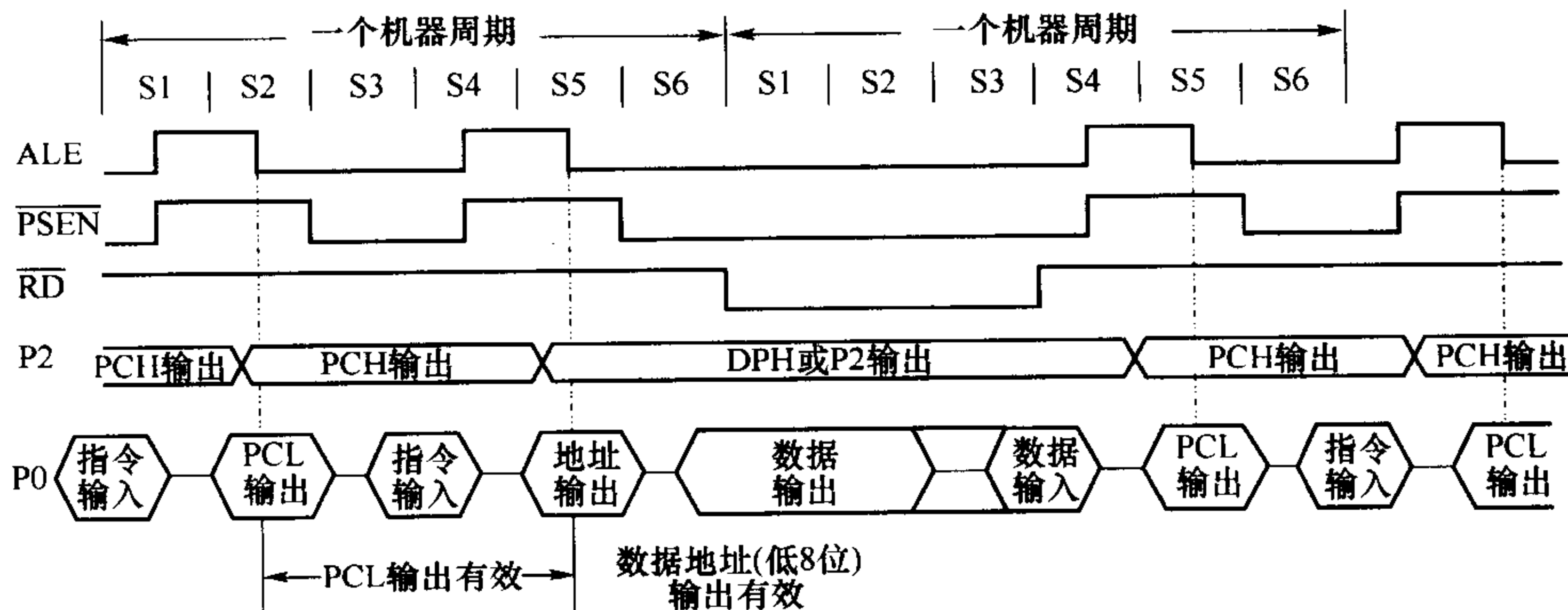
(6) 至此,由一个机器周期的第 3 个振荡周期(S2P1)开始到第 8 个振荡周期(S4P2)结束,完成了一次对外部程序存储器的访问过程。

由图 3.3.2 可以看出,在一个机器周期内地址锁存允许信号 ALE 出现了 2 个脉冲,外部程序存储器读选通信号  $\overline{\text{PSEN}}$  也出现了 2 个脉冲,因此在一个机器周期内单片机可

以访问 2 次外部程序存储器。



(a) 执行非MOVX类指令时



(b) 执行MOVX类指令时

图 3.3.2 外部程序存储器的操作时序

当系统中接有外部数据存储器,执行 MOVX 指令时,外部程序存储器的操作时序有所变化。原因在于,执行 MOVX 指令时,16 位地址应转而指向数据存储器。在指令读入以前,P2 口、P0 口输出的地址 PCH、PCL 指向程序存储器。在指令读入并判断是 MOVX 指令后,在该机器周期的 S5 状态,ALE 锁存的 P0 口的地址不是程序存储器的低 8 位,而是数据存储器的地址。若执行的是 MOVX A,@DPTR 或 MOVX @DPTR,A 指令,则此地址就是数据指针的低 8 位 DPL;同时,在 P2 口上出现的是数据指针的高 8 位 DPH。若执行的是 MOVX A,@Ri 或 MOVX @Ri,A 指令,则 Ri 的内容为低 8 位地址,而 P2 口线上将是 P2 口锁存器的内容。在同一机器周期中将不再出现 PSEN 有效的取指信号,下一个机器周期中 ALE 的有效锁存信号也不再出现;而当 RD(或 WR)有效时,在 P0 口线上将出现有效的输入数据(或输出数据)。从时序图 3.3.2 可以看出:

(1) 将 ALE 作为定时脉冲输出时,执行一次 MOVX 指令其会丢失 1 个脉冲;

(2) 只有执行 MOVX 指令时的第 2 个机器周期期间,地址总线才由数据存储器使用。

### 3.3.2 外部数据存储器的访问

#### 1. 扩展外部数据存储器的连接方法

AT89S52 单片机在芯片内已经集成了 256 B 的数据存储器,当应用系统的控制要求比较简单、需要处理的数据量不大时,这 256 B 的存储空间基本够用。但当应用系统的控制功能比较复杂、需要处理的数据量较大时,可向外扩展数据存储器,最大可扩展到 64 KB 存储空间。

AT89S52 单片机的外部数据存储器扩展方法如图 3.3.3 所示。

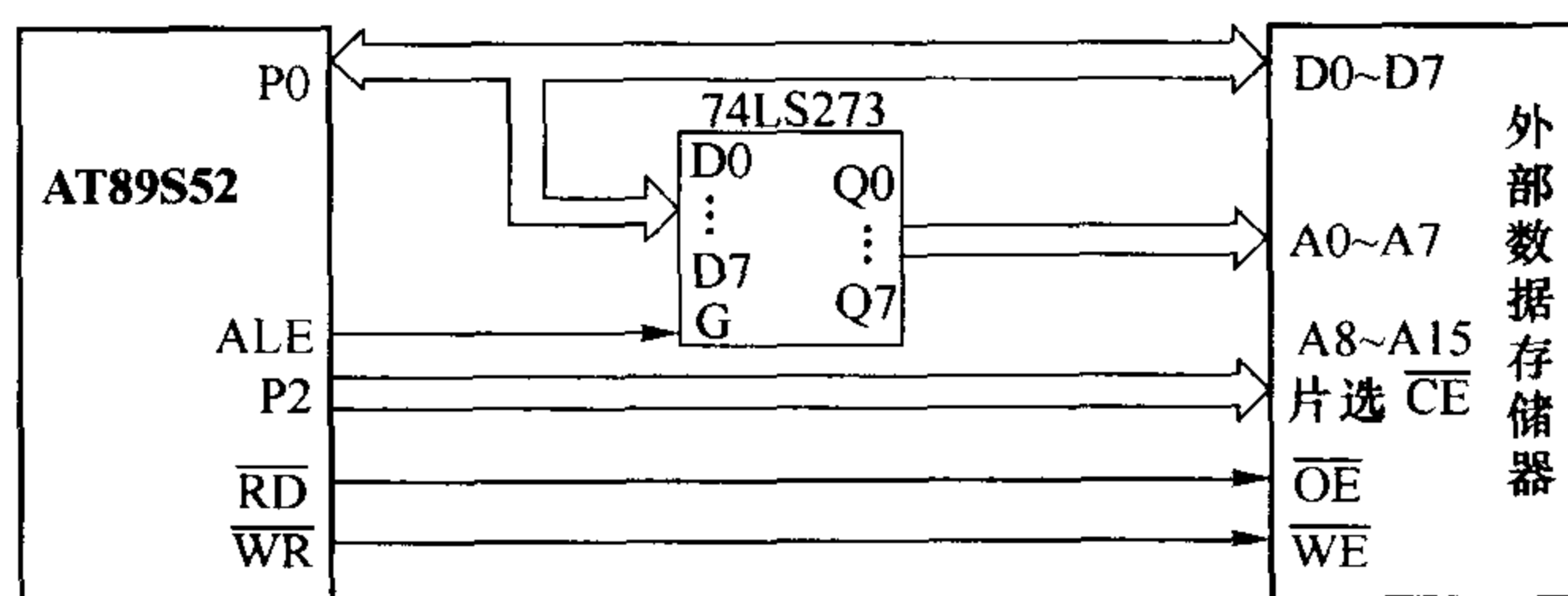


图 3.3.3 外部数据存储器扩展

虽然外部数据存储器 and 外部程序存储器共用 0000H~FFFFH 的 64 KB 地址空间,但两者的读写控制信号不同,外部数据存储器的读和写分别由  $\overline{\text{RD}}$ /P3.7 和  $\overline{\text{WR}}$ /P3.6 信号控制,外部程序存储器的读选通由信号  $\overline{\text{PSEN}}$  控制,因此不会发生地址重叠的现象。

访问外部数据存储器使用 MOVX 类指令,如 MOVX A, @Ri; MOVX @Ri, A; MOVX A, @DPTR; MOVX @DPTR, A。

#### 2. 访问外部数据存储器的时序

图 3.3.4 为访问外部数据存储器的读时序。

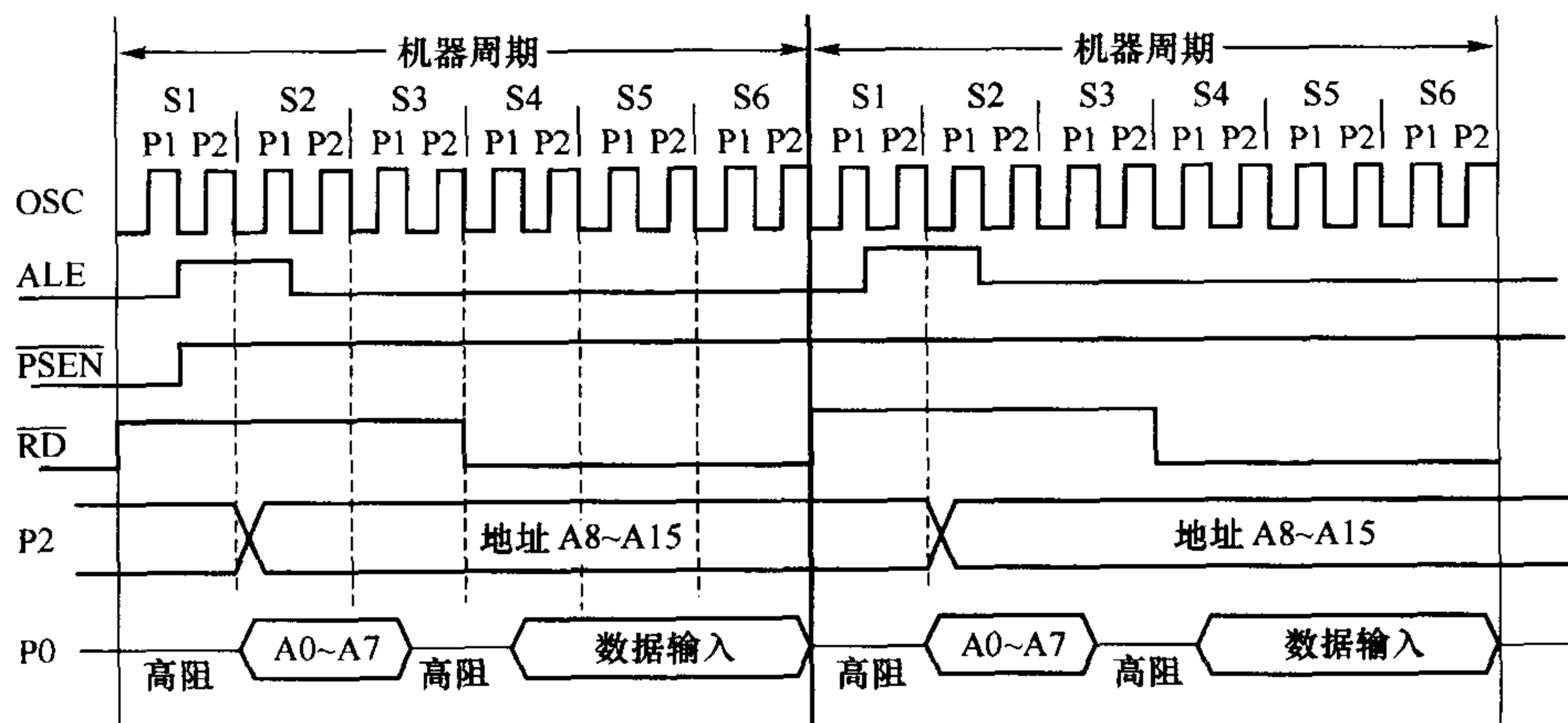


图 3.3.4 外部数据存储器的读时序



具体过程分析如下。

(1) S1 状态,地址锁存允许信号 ALE 的上升沿(低电平变为高电平),开始读周期。

(2) S2 状态,CPU 把低 8 位地址送到 P0 口线,同时把高 8 位地址送到 P2 口线(若执行 MOVX @DPTR 类指令),地址锁存允许信号 ALE 的下降沿(高电平变为低电平)时,将低 8 位地址信息锁存到外部地址锁存器中,而高 8 位地址信息此后一直锁存在 P2 口,无须再外加锁存器。

(3) S3 状态,P0 口总线进入到高阻状态。

(4) S4 状态,读控制信号  $\overline{RD}$  变成有效,经一定的延时将被寻址的外部数据存储器指定单元的数据送至 P0 口数据总线。

(5) S6P2 状态结束时, $\overline{RD}$  信号的上升沿到来,被寻址的外部数据存储器的总线驱动器悬空,P0 口总线又进入到高阻状态。

(6) 至此,由一个机器周期的第 2 个振荡周期(S1P2)开始到一个机器周期的第 12 个振荡周期(S6P2)结束,完成了一次对外部数据存储器的读操作过程。

使用 MOVX @DPTR, A 指令对外部数据存储器进行写操作的时序如图 3.3.5 所示。

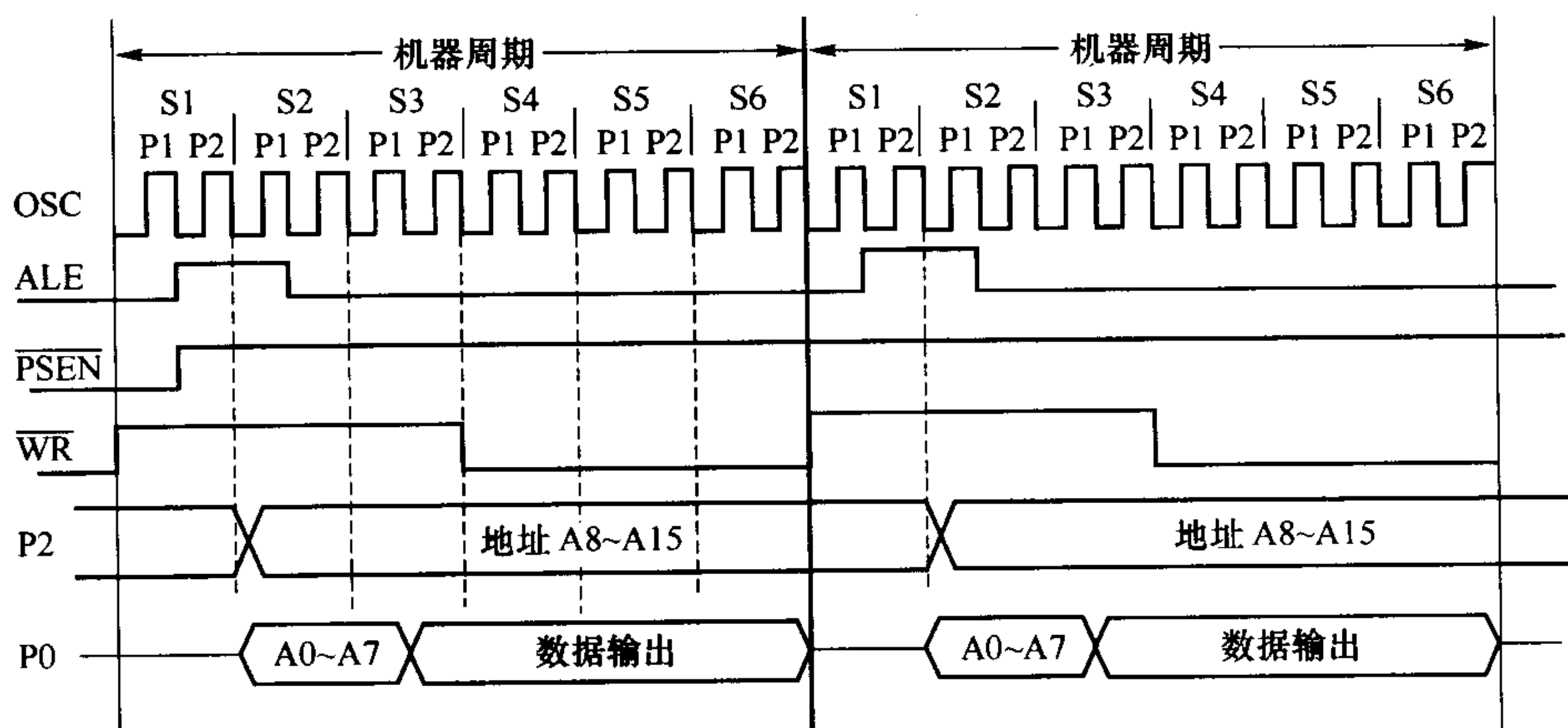


图 3.3.5 外部数据存储器写操作的时序

将外部数据存储器写操作时序与外部数据存储器读操作时序比较可以看出,写操作时序与读操作时序基本相同,但有所区别,两者的区别如下。

(1) 写操作过程是 CPU 主动将数据送至 P0 口数据总线,而读操作则是由外部数据存储器读入数据后送至 P0 口数据总线。

(2) CPU 向 P0 口数据总线送上低 8 位地址并被锁存到地址锁存器后,在 S3 状态就可以向 P0 口总线送上要写入的数据;在 S3P2~S4P2 区间,P0 口总线不再呈现高阻状态。

(3) 在 S4 状态,写控制信号  $\overline{WR}$  低电平有效,选通被寻址的外部数据存储器单元,经

片刻的延时后将 P0 口数据总线上的数据写入到被寻址的外部数据存储器单元中,在 S6P2 结束前完成对外部数据存储器的写操作。

### 3.4 片内 Flash 存储器操作

片内 Flash 操作包括对 Flash 标志字节的读出、并行编程、串行编程、程序加密等。可以利用计算机、单片机等设备实现对 Flash 存储器的操作。

#### 3.4.1 标志字节

所谓标志字节是 Flash 存储器的生产厂商在生产 AT89S 系列单片机时,写入到 Flash 存储器中的一组用以说明单片机的生产厂商、型号和编程电压等的特征信息。

在单片机的封装外壳上,会以某种形式印刷这组信息。之所以将这组信息以标志字节的形式存储在 Flash 存储器中,是为了在所印刷的信息被磨损后可以通过读出标志字节内容来获得这组信息,方便使用。

AT89S52 的标志字节共有 3 B,具体在存储器中的地址和含义如表 3.4.1 所示。

表 3.4.1 标志字节的地址、内容和代表的含义

地 址	内 容	代表的含义
000H	1EH	表示生产厂商为 ATMEL 公司
100H	51H	表示为 AT89S51 型单片机
	52H	表示为 AT89S52 型单片机
200H	06H	—

通用编程器即通过读标志字节识别所编程单片机的生产厂商、型号等信息,以便进行编程电压的控制。

#### 3.4.2 程序存储器的加密

为了保护所存储的程序的安全性,防止被非法读出,保护开发者的合法利益,需要对写入 Flash 存储器中的程序进行加密。AT89S 系列单片机提供了较强的加密功能,可以对 Flash 存储器实施不同程度的封锁,以阻止对程序的非法读出,保护程序的安全。

AT89S 系列单片机提供了 3 位加密位 LB1、LB2 和 LB3,对每位加密位可维持原来的非编程状态(U),也可进行编程(P),根据每位加密位是否进行了编程便可组合形成几种不同的保护模式,如表 3.4.2 所示。

表 3.4.2 程序加密位的保护模式

模式	加密位			组合加密功能
	LB1	LB2	LB3	
1	U	U	U	没有程序加密功能
2	P	U	U	禁止在外部程序存储器中执行 MOVC 类指令读取内部程序存储器中的指令代码; $\overline{EA}$ 被采样并在复位时被锁存; 禁止对 Flash 存储器再编程
3	P	P	U	同模式 2, 并禁止内部存储器的校验
4	P	P	P	同模式 3, 并禁止外部存储器的执行

表 3.4.2 中未列出的其他组合方式未被定义。从表 3.4.2 中可以看出, 当 LB1 被编程时,  $\overline{EA}$  引脚上的信号(电平)被采样并在复位时被锁存。如果程序锁定位被编程后一直没有复位操作, 则锁存器中的值是随机的, 直到复位后起作用。

对程序存储器加密需要根据所希望采取的加密保护模式对 3 位加密位 LB1、LB2 和 LB3 进行编程。编程按照 LB1→LB2→LB3 的顺序按位进行。注意, 在对各位加密位进行编程时, 其控制信号不同, 通过并行编程的方法对加密位进行编程(参见 3.4.3 小节)。

### 3.4.3 Flash 存储器的并行编程

Flash 存储器最突出的优点之一是可多次擦写且操作简单, 无须专用设备, 这大大地方便了实际应用, 用户可以非常方便地修改程序存储器中的程序。

#### 1. Flash 编程器的并行编程方式

AT89S52 单片机的内部 Flash 存储器出厂时处于可编程状态, 除标志字节已经有存储数据外, 其他存储单元的内容均为 FFH。编程时需要接 12 V 编程电压, 与通用的 Flash 编程器或 EPROM 编程器兼容, 因此可使用常规的 Flash 编程器或 EPROM 编程器进行编程。AT89S52 单片机的编程以字节为单位, 逐位编程。图 3.4.1 为并行编程的接口电路图。

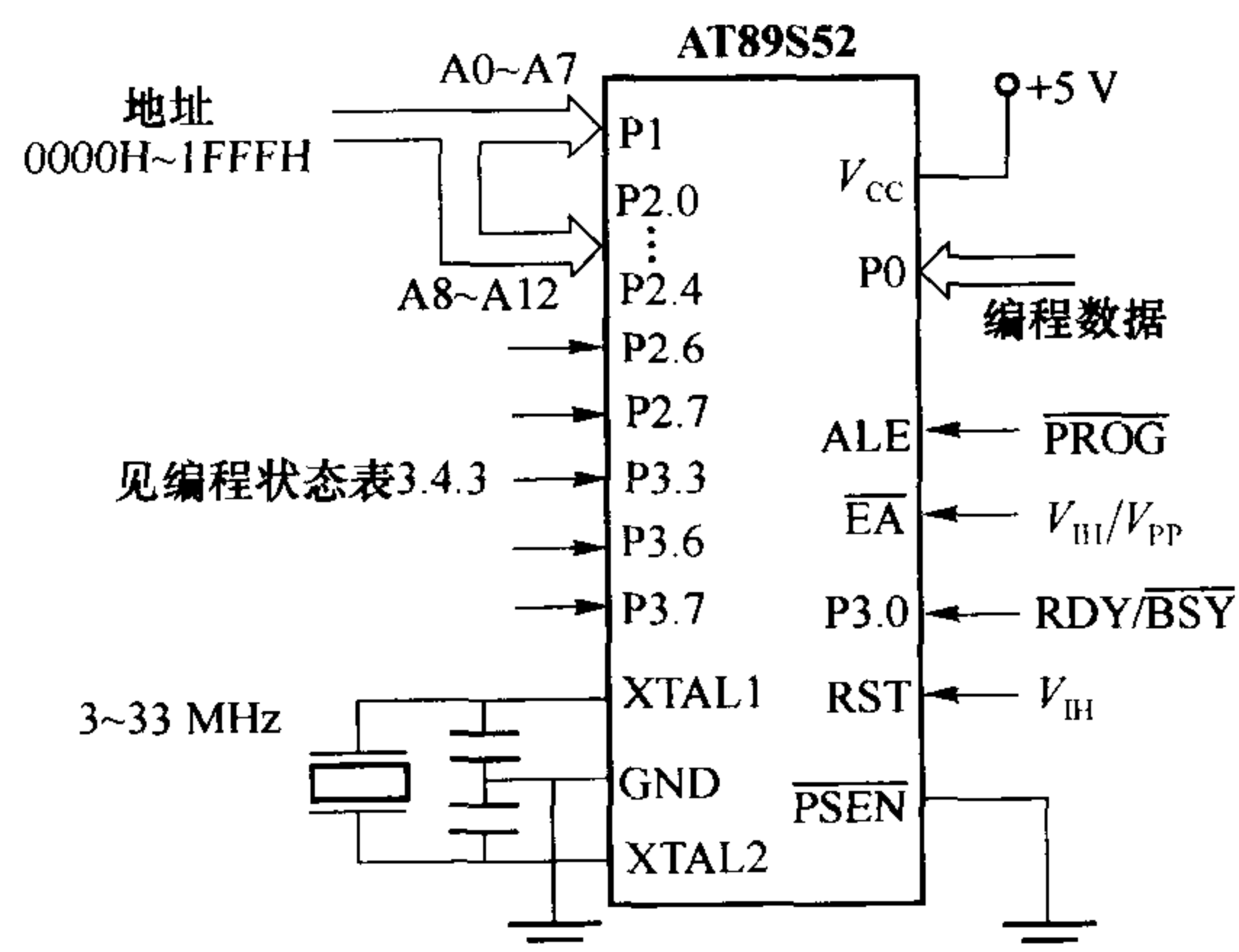


图 3.4.1 AT89S52 Flash 存储器并行编程的接口电路

## 2. 并行编程的算法

从编程接口电路可以看出,AT89S52 单片机编程时,除地址线和数据线外还需要一些控制信号,表 3.4.3 为编程时这些控制信号的状态情况。

表 3.4.3 AT89S52 Flash 存储器编程状态

模 式	RST	$\overline{\text{PSEN}}$	ALE/ $\overline{\text{PROG}}$	$\overline{\text{EA}}/\text{V}_{\text{PP}}$	P2.6	P2.7	P3.3	P3.6	P3.7	P0.7~P0.0 数据	P2.4 ~ P2.0	P1.7~P1.0
											地址	
写代码数据	H	L	负脉冲 <sup>②</sup>	12 V	L	H	H	H	H	Din	A12~A8	A7~A0
读代码数据	H	L	H	H	L	L	L	H	H	Dout	A12~A8	A7~A0
写加密位 LB1	H	L	负脉冲 <sup>③</sup>	12 V	H	H	H	H	H	×	×	×
写加密位 LB2	H	L	负脉冲 <sup>③</sup>	12 V	H	H	H	L	L	×	×	×
写加密位 LB3	H	L	负脉冲 <sup>③</sup>	12 V	H	L	H	H	L	×	×	×
读加密位 1,2,3 <sup>④</sup>	H	L	H	H	H	H	L	H	L	P0.2 P0.3 P0.4	×	×
芯片擦除	H	L	负脉冲 <sup>①</sup>	12 V	H	L	H	L	L	×	×	×
读标志字节	H	L	H	H	L	L	L	L	L	1EH	×0000	00H
	H	L	H	H	L	L	L	L	L	52H	×0001	00H
	H	L	H	H	L	L	L	L	L	06H	×0010	00H

注:① 片擦除 $\overline{\text{PROG}}$ 脉宽 200~500 ns;

② 写代码数据  $\overline{\text{PROG}}$ 脉宽 200~500 ns;

③ 写加密位  $\overline{\text{PROG}}$ 脉宽 200~500 ns;

④ 读加密位 LB1、LB2、LB3 分别出现在 P0.2、P0.3 和 P0.4 引脚;

⑤ 编程期间 RDY/ $\overline{\text{BSY}}$ 信号从 P3.0 引脚输出;

⑥ ×表示不关心。

AT89S52 单片机内部 Flash 为 8 KB,地址范围为 0000H~1FFFH,因此编程时需要 13 位地址线。编程时,被编程存储单元的地址由 P1 口和 P2 口的 P2.0~P2.4 输入(13 位地址),编程代码从 P0 口输入,P2.6、P2.7、P3.3、P3.6 和 P3.7 引脚的电平依据表 3.4.3 设置。RST 引脚接高电平, $\overline{\text{PSEN}}$ 引脚接低电平(接地),ALE/ $\overline{\text{PROG}}$ 引脚接编程负脉冲,每次写入代码的脉冲宽度为 200~500 ns, $\overline{\text{EA}}/\text{V}_{\text{PP}}$ 是编程电压的输入引脚,按规定要求接 12 V 编程电压,编程时的振荡频率为 3~33 MHz。

并行编程按以下步骤进行:

- (1) 地址线 P1.0~P1.7、P2.0~P2.4 上输入需要的单元地址;
- (2) 数据线 P0.0~P0.7 上输入待写入的字节代码数据;
- (3) 使各控制信号有效;
- (4)  $\overline{\text{EA}}/\text{V}_{\text{PP}}$ 引脚编程电压上升至 12 V;
- (5) ALE/ $\overline{\text{PROG}}$ 引脚施加编程负脉冲,每个编程脉冲写入一个字节或一个加密位,

写入周期由单片机内部自动设定,其典型值为  $50\ \mu\text{s}$ 。

(6) 重复步骤(1)~(5),改变所输入的地址和代码数据,一直到所有的程序写完为止。

### 3. 数据查询方式

AT89S52 可以通过数据查询的方式来判断一个编程的写周期是否结束。在写周期内,如果想读出最后写入字节的数据,则读出的数据的最高位(在 P0.7 引脚)是写入数据的反码。

写周期一结束,写入该字节的正确数据将出现在数据总线上,标志着下一个写周期的开始。数据查询可在写周期后的任何时刻进行。

### 4. 准备就绪/忙(RDY/ $\overline{\text{BSY}}$ )信号

数据查询方式实际上是提供给编程者监视编程过程的一种方法。除此之外,还可在编程过程中通过准备就绪/忙(RDY/ $\overline{\text{BSY}}$ )信号对编程过程进行监视。

在编程过程中,当 ALE/ $\overline{\text{PROG}}$  引脚上的负脉冲由低变高后,P3.0 引脚(RDY/ $\overline{\text{BSY}}$ )上的电压被拉低,表示正处于编程状态(忙)。当编程结束后,P3.0 引脚的电平被拉高,表示前一字节的编程已经完成,处于准备就绪状态,可以开始下一个写周期。

### 5. 程序的校验

若 Flash 程序存储器的加密位没有被编程,则可以通过地址线和数据线读出 Flash 程序存储器中的程序,对其进行校验。

图 3.4.2 为 AT89S52 片内 Flash 存储器并行校验的接口电路。

对比并行编程接口电路图 3.4.1 可看出,地址仍由 P1 口和 P2 口的 P2.0~P2.4 输入(13 位地址),校验的程序代码从 P0 口输出,P2.6、P2.7、P3.3、P3.6 和 P3.7 引脚的电平仍然依据表 3.4.3 设置,在  $V_{\text{CC}}$  引脚和 P0 口的各位端接  $10\ \text{k}\Omega$  的上拉电阻。

需要注意的是,程序加密位不能按照上述办法进行直接校验,而要通过观察它们的加密功能是否被实现来判断。

### 6. Flash 程序存储器的片擦除

在并行编程模式下对 AT89S52 内部 Flash 存储器芯片的整片擦除,各控制信号按照表 3.4.3 中芯片擦除一栏的值进行设置。

擦除后 Flash 存储器除标志字节外其他存储单元的内容均为 FFH。在对 Flash 存储器重新编程前,需要先执行芯片的擦除操作。

## 3.4.4 Flash 存储器的串行编程

### 1. Flash 存储器的串行编程方式

当 RST 引脚接高电平时,可通过串行接口 ISP 对 AT89S52 Flash 进行编程。串行接口 ISP 由引脚 P1.5/MOSI、P1.6/MISO 和 P1.7/SCK 组成,P1.5/MOSI 作为串行指令的输入引脚,P1.6/MISO 为串行数据的输出引脚,P1.7/SCK 为串行移位脉冲的输入

引脚。串行编程/下载接口电路如图 3.4.3 所示。

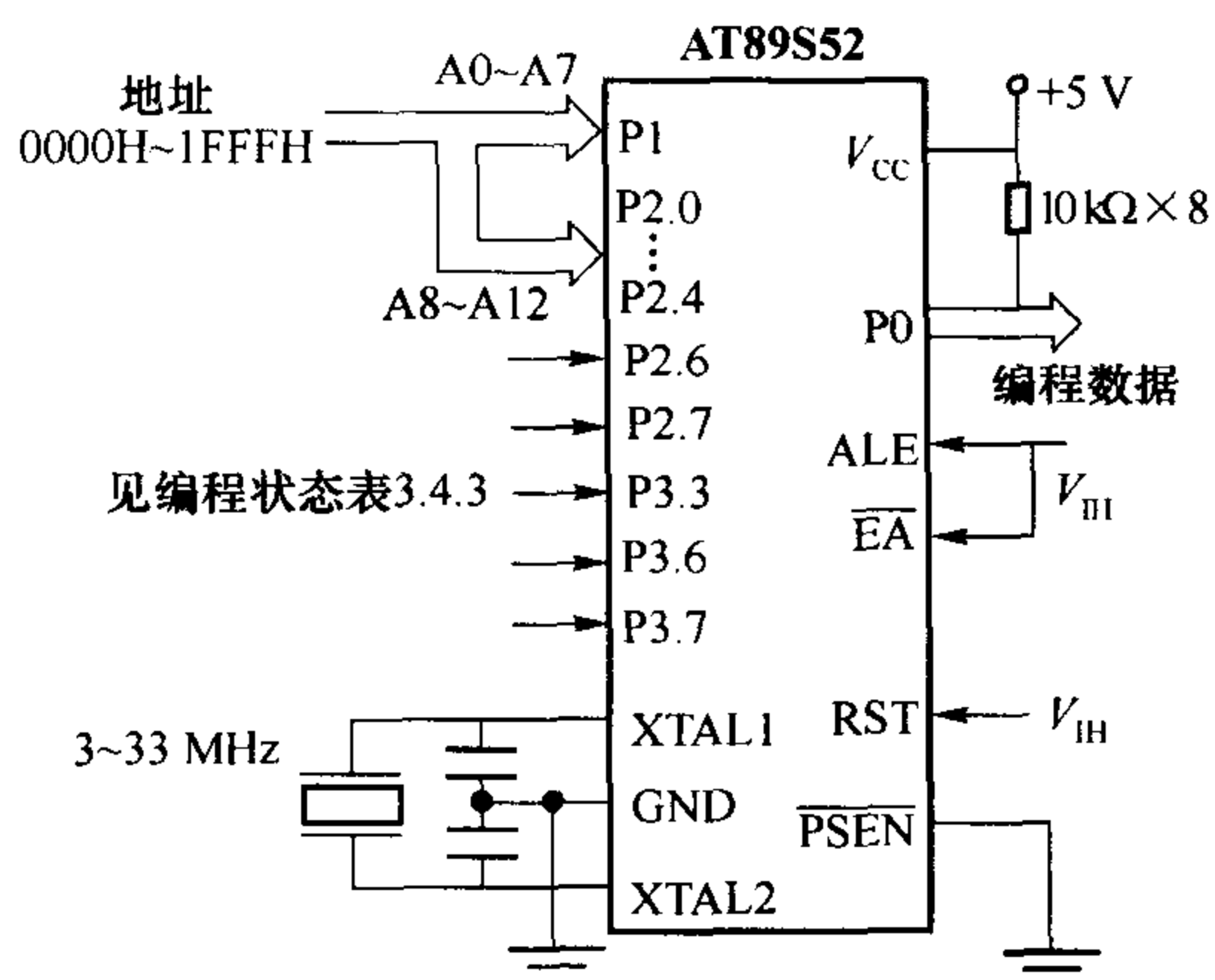


图 3.4.2 AT89S52 Flash 存储器并行校验接口电路

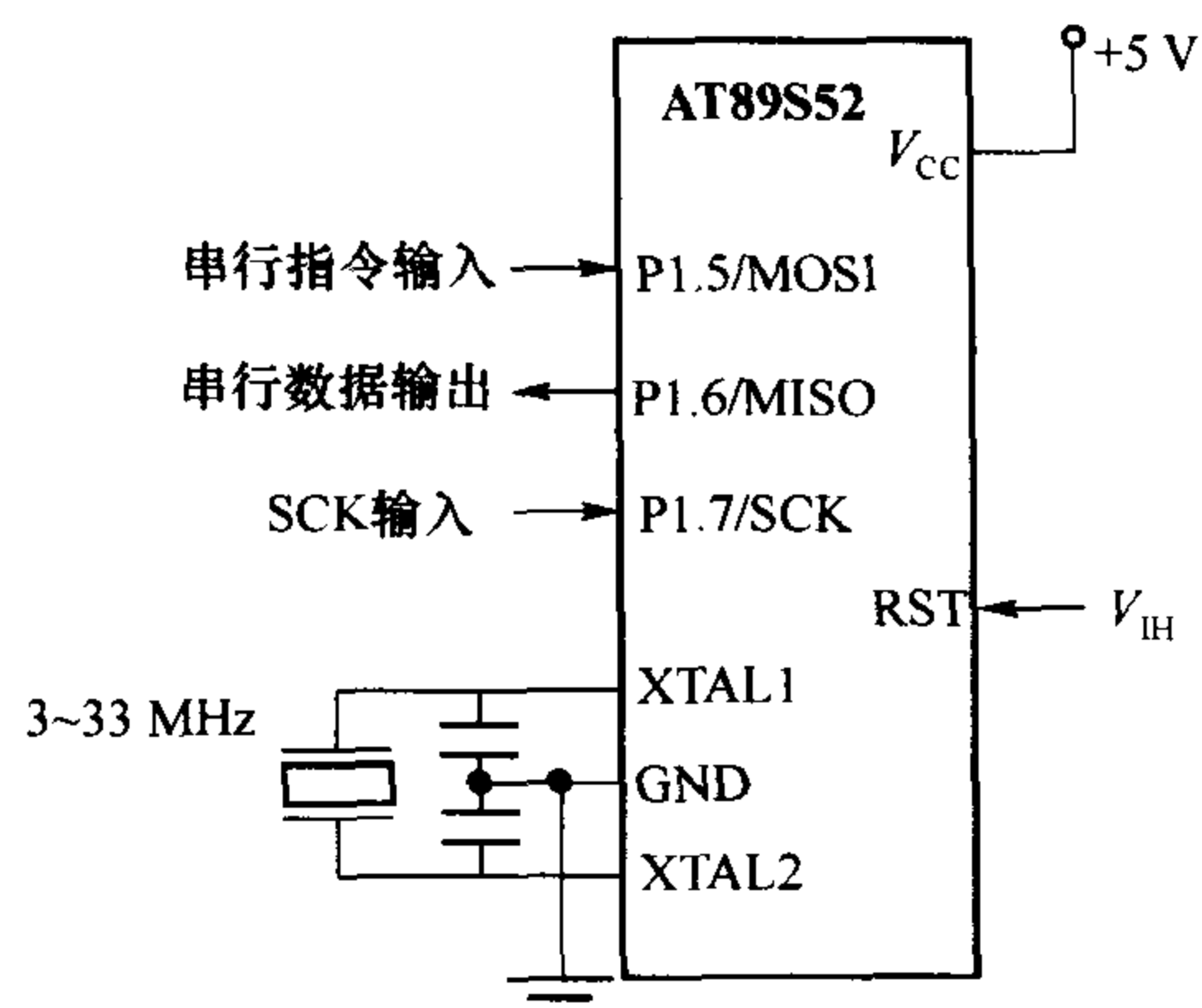


图 3.4.3 AT89S52 Flash 存储器串行编程/下载接口电路

在 RST 被置成高电平之后、执行串行编程操作之前,必须先执行串行编程允许指令,这样才可以实现串行编程。如果是对 Flash 重新编程,则必须先执行片擦除操作,擦除后 Flash 存储器芯片除标志字节外其他存储单元的内容均为 FFH。

在串行编程工作模式下时钟振荡器有外部时钟信号和芯片内时钟发生电路两种方式。采用外部时钟信号方式时,外部时钟信号由 XTAL1 引脚输入,XTAL2 引脚悬空;采用芯片内时钟发生电路方式时,在 XTAL1 和 XTAL2 引脚之间跨接晶振和微调电容。

无论采用哪一种时钟方式,串行移位脉冲 SCK 均应低于晶振频率的 1/16。如晶振频率为 33 MHz,则 SCK 的最高频率应该是 2 MHz。

## 2. Flash 存储器的串行编程算法

按照下列步骤对 Flash 实现串行编程。

(1) 对 RST、 $V_{CC}$  和 GND 引脚加电,加电次序如下:

- ① 在  $V_{CC}$  和 GND 引脚之间加电源电压;
- ② 将 RST 设置为高电平(若采用外部时钟信号,则必须延时 10 ms 后方可)。

(2) 在 P1.5/MOSI 引脚输入编程允许指令。

(3) 在 P1.5/MOSI 引脚输入写程序存储器指令。

AT89S52 的串行编程指令中包含了编程单元地址和代码数据,向 P1.5/MOSI 引脚输入写程序存储器指令时,便确定了可编程的字节地址和指令数据。写入周期采用内部自动定时的方式,在  $V_{CC}=5\text{ V}$  时其典型值不大于 1 ms。

编程可按字节模式或页模式写入。在采用字节编程模式时,编程的地址单元和代码数据包含在指令的第 2、3、4 字节中。

(4) 读指令。使用读指令,在 P1.6/MISO 引脚上读出芯片内 Flash 程序存储器任意存储单元中的内容,用于编程校验。

(5) 编程结束后将 RST 引脚置低电平,系统恢复到正常操作状态。

如果需要,可按照下面的步骤实施断电:



- (1) 将 XTAL1 引脚置成低电平(若使用外部时钟);
- (2) 将 RST 引脚置低电平;
- (3) 关断电源  $V_{CC}$ 。

### 3. AT89S52 的串行编程指令

从图 3.4.3 可以看到,AT89S52 单片机串行编程的接口电路较并行编程的接口电路要简单许多,只需要 P1.5/MOSI 作为串行指令的输入引脚,P1.6/MISO 作为串行数据的输出引脚,P1.7/SCK 作为编程时钟的输入引脚即可,编程的控制功能主要靠软件来实现,输入不同的编程指令便可实现不同的编程操作。

AT89S52 单片机串行编程指令为 4 B 格式,表 3.4.4 给出了 AT89S52 单片机串行编程指令的格式构成和各指令的编码。

表 3.4.4 AT89S52 单片机串行编程指令集

指 令		指令格式			
		字节 1	字节 2	字节 3	字节 4
编程允许	写/擦允许	1010 1100	0101 0011	×××× ××××	×××× ××××
	读允许	1010 1100	0101 0011	×××× ××××	0110 1001
片擦除		1010 1100	100× ××××	×××× ××××	×××× ××××
读程序(字节模式)		0010 0000	×××A12 A11~A8	A7~A4 A3~A0	D7~D4 D3~D0
写程序(字节模式)		0100 0000	×××A12 A11~A8	A7~A4 A3~A0	D7~D4 D3~D0
写加密位		1010 1100	1110 00B1B2	×××× ××××	×××× ××××
读加密位		0010 0100	×××× ××××	×××× ××××	×××LB3 LB2LB1××
读程序(页模式)		0011 0000	×××A12 A11~A8	字节 0	字节 1~字节 255
写程序(页模式)		0101 0000	×××A12 A11~A8	字节 0	字节 1~字节 255

现对表 3.4.4 说明如下。

- (1) 在输入串行编程允许指令的同时,RST 引脚置成高电平。
- (2) 以字节模式读/写程序存储器的指令中,第 2、3 字节的内容为存储单元的地址,第 4 字节为代码数据。
- (3) 以页模式读/写程序存储器的指令中,第 2 字节的内容为页地址,第 3 字节为字节代码的字节 0,第 4 字节为字节代码的字节 1,连续输入/输出字节代码,直到字节代码的字节为 255。256 字节为 1 页。
- (4) 在写模式下,字节代码或页代码输入一结束,便立刻进入写周期的内部自动定时。
- (5) 在写程序加密位时,数位 B1 和 B2 的值与保护模式的关系如表 3.4.5 所示。

表 3.4.5 B1 和 B2 的值与保护模式的关系

B1	B2	保护模式	说 明
0	0	1	无程序加密功能
0	1	2	LB1 有效
1	0	3	LB2 有效
1	1	4	LB3 有效

关于保护模式及 LB1、LB2 和 LB3 的意义见表 3.4.2。

(6) 在读出程序加密位时,加密位 LB3、LB2 和 LB1 顺序出现在串行数据输出口 P1.6/MISO 的 D4、D3 和 D2 位上。

#### 4. 串行编程模式下的数据查询

AT89S52 在串行编程模式下也具有数据查询功能。

在写周期内,读出最后写入的字节时,则在串行数据输出口 P1.6/MISO 引脚上出现写入字节数据最高位的反码。

图 3.4.4 为串行编程模式下的时序图,表 3.4.6 为时序参数。

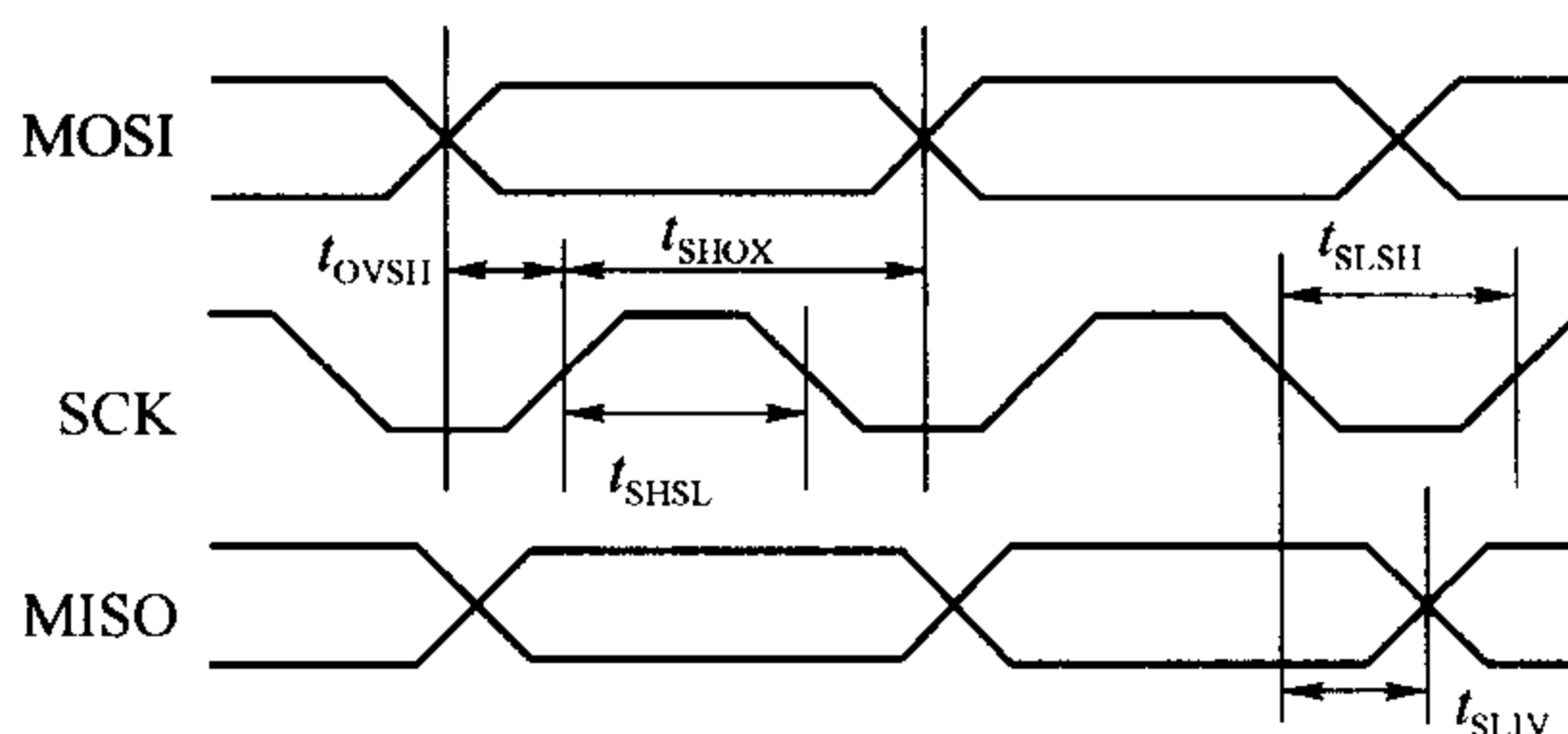


图 3.4.4 串行编程模式下的时序图

表 3.4.6 Flash 编程串行模式时序参数

序 号	参 数	符 号	最小值	典型值	最大值	单 位
1	振荡器频率	$1/t_{CLCL}$	0		33	MHz
2	振荡器周期	$t_{CLCL}$	30			ns
3	SCK 高电平宽度	$t_{SHSL}$	$2t_{CLCL}$			ns
4	SCK 低电平宽度	$t_{SLSH}$	$2t_{CLCL}$			ns
5	MOSI 有效到 SCK 脉冲上升沿	$t_{OVSH}$	$t_{CLCL}$			ns
6	SCK 上升沿到 MOSI 有效	$t_{SHOX}$	$2t_{CLCL}$			ns
7	SCK 下降沿到 MISO 有效	$t_{SLIV}$	10	16	32	ns
8	片擦除指令执行周期	$t_{EARSE}$			500	ms
9	串行字节写周期时间	$t_{SWC}$			$64t_{CLCL} + 400$	$\mu s$

## 习 题

1. AT89S52 的数据存储器与 AT89C51、MCS-51 等单片机的数据存储器在结构上有哪些不同?
2. 简述 AT89S52 的数据存储器的地址分配和配置。
3. 说明 AT89S52 片内 RAM 存储空间分布情况。
4. AT89S52 的特殊功能寄存器 SFR 有哪些? 这些寄存器占用了哪些地址空间?
5. AT89S52 的哪些寄存器可按位寻址? SFR 中共有多少个可按位寻址单元? 内部

RAM中共有多少个可按位寻址单元?

6. 怎样选择双数据指针寄存器 DPTR0 和 DPTR1?
7. 了解外部数据存储器的访问过程。
8. 地址锁存器的作用是什么?
9. 分析访问外部程序存储器的时序。
10. 分析访问外部数据存储器的时序。
11. Flash 存储器与 EPROM 和 EEPROM 相比,有哪些突出的优点?
12. 了解各类半导体存储器的基本特点。
13. 何谓 AT89S52 的标志字节? 简述读出标志字节中内容的步骤。
14. 为什么要对程序存储器实施加密? AT89S52 有哪几种程序存储器加密功能?
15. 简述 AT89S52 程序存储器加密操作的方法。
16. 何谓程序存储器的并行编程和串行编程?
17. 在对 AT89S52 程序存储器并行编程时,有哪些控制信号? 这些控制信号是怎样变化的或其状态是什么?
18. 简述 AT89S52 程序存储器并行编程的算法。
19. 为什么要对程序存储器中所存储的程序进行校验? 在进行并行校验时,图 3.4.2 中端接在 +5 V 电源和 P0 引脚之间的 10 k $\Omega$  电阻的作用是什么?
20. 简述 AT89S52 程序存储器串行编程的算法。
21. 简述 AT89S52 程序存储器串行编程指令的构成及指令各字节的意义。

## 第 4 章 AT89S52 指令系统

AT89S52 单片机硬件资源比较丰富,但只有硬件资源和软件程序的完美配合才能使单片机系统能够按照控制功能的要求工作,所以必须编写一系列的程序,通过其执行以完成指定的任务。程序是按系统功能要求编排的一系列指令的集合。

指令是单片机执行操作的命令,所有指令的集合称为指令系统。

指令有两种描述形式:用机器语言描述的指令和用汇编语言描述的指令。机器语言指令用二进制编码表示,是 CPU 唯一能直接识别和执行的指令。汇编语言指令是一种符号指令,由助记符、符号和数字等来表示指令,与机器语言指令一一对应。

机器语言编写的程序不便于记忆、查错和修改,汇编语言比机器语言容易理解和记忆。

近年来,适用于单片机的高级语言已日渐成熟,应用于 51 系列单片机编程的高级语言主要为 PLM、BASIC、C 语言等。高级语言用自然语言描述,使用高级语言编写的程序具有效率高、可读性好和可移植性强的特点。实际编写程序时通常使用汇编语言和高级语言。

汇编语言和高级语言编写的程序都不能直接被 CPU 执行,需要使用相应的软件编译,并生成目标代码,即用十六进制或二进制表示的机器码,再利用编程器、烧写器或通过串口等将目标代码下载到单片机的 Flash 或程序存储器中,单片机才能执行并完成相应的功能。目标代码生成的流程如图 4.1 所示。

本章按标准汇编指令来分析指令系统的功能和使用方法。AT89S52 单片机指令与 MCS-51 指令集兼容,共有 111 条指令,其中单字节指令 49 条、双字节指令 45 条、三字节指令 17 条。单字节指令对应的机器码占 1 B,双字节指令对应的机器码占 2 B,三字节指令对应的机器码占 3 B。按照指令的执行时间分类,单周期指令 64 条、双周期指令 45 条、四周期指令 2 条(乘除指令)。指令的周期数越多,对应指令的执行时间越长,单周期指令的执行时间最短,双周期指令次之。编写程序时,应尽量使用周期数少的指令,以加快程序的执行速度,节省执行时间,并减少程序“跑飞”的机会。

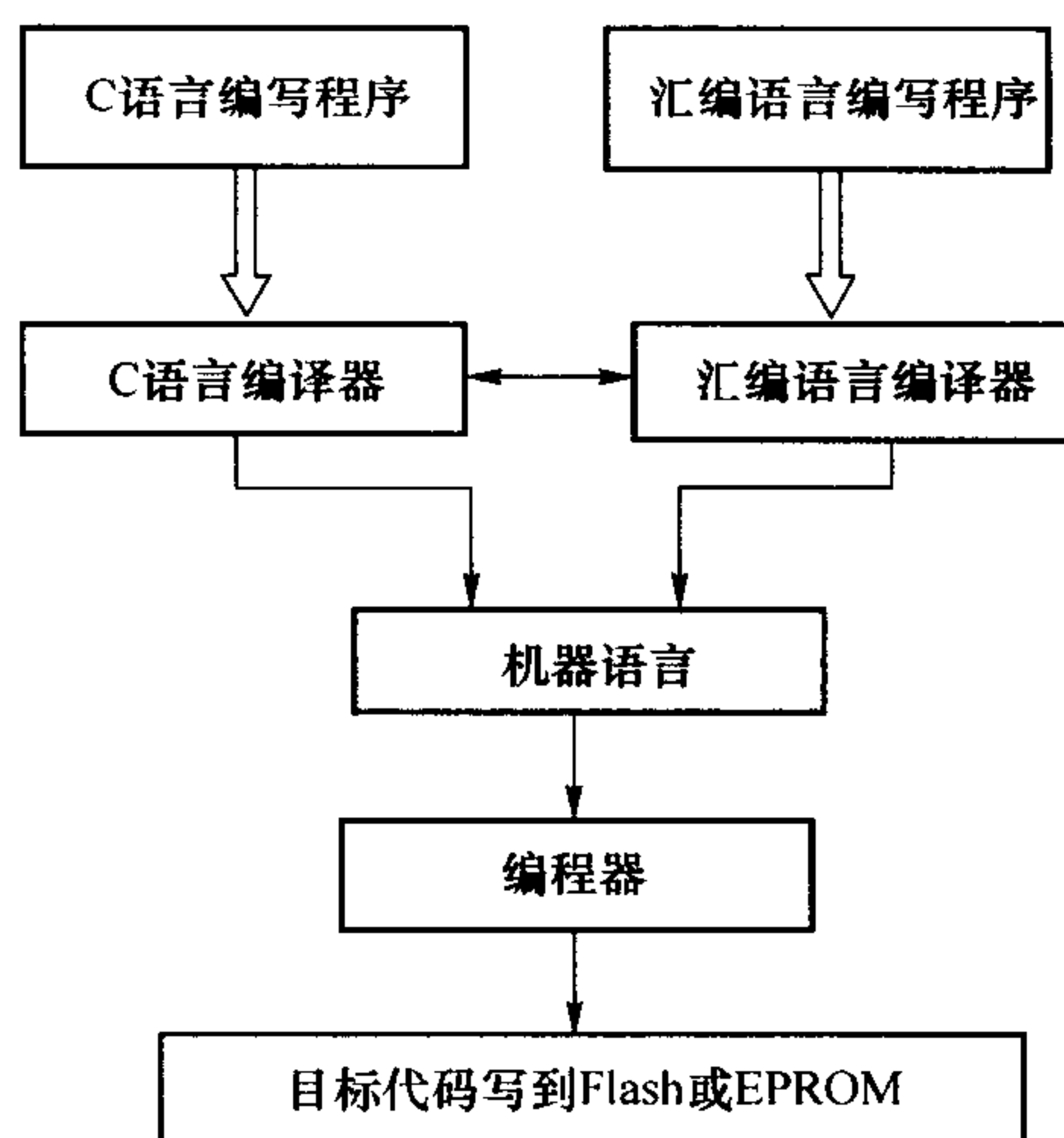


图 4.1 目标代码生成过程

## 4.1 汇编语言指令格式

汇编指令分为两类:执行指令和伪指令。执行指令即指令系统给出的各种指令;伪指令由汇编程序规定,是提供汇编控制信息的指令。

### 4.1.1 汇编语言执行指令格式

MCS-51 单片机汇编语言执行指令的格式如下:

格式: [标号:] 操作码[操作数 1],[操作数 2];注释

标号(可以没有)是用户定义的符号。以字母开始,后跟 1~8 个英文字母或数字,并以冒号“:”结尾。如“DELAY:MOV A, #08H;”代表本条语句的标号为 DELAY,程序中调用或跳转时直接利用标号即可。标号名称尽量用与该段程序内容相关且有意义的英文单词或汉语拼音等,如延时程序用 DELAY,多字节加法用 MULTIADD 等。标号的实际意义代表当前语句在程序存储器中的存放地址,如 0100H,作为程序跳转或转移的标记,该地址编译软件会自动产生。在机器语言或编译后的目标代码中没有 DELAY 这样的标号,只有标号对应的地址。

操作码也称助记符,汇编语言中由英文单词缩写而成,反映指令的功能,如 MOV、ADD 等。

操作数(可以没有)是指参加操作的数据或数据存放的地址。不同功能的指令可以有 3 个、2 个、1 个或者根本没有操作数,与操作码之间至少需要用一个空格隔开。反映指令的操作对象。指令中操作数 1 称为目的操作数,操作数 2 称为源操作数。

注释(可以没有)是指程序员对该条指令或程序段的说明,通常对程序的功能、主要内容、进入和退出子程序的条件等进行注释,以提高程序的可读性。汇编时不被编译,因而在机器代码的目标程序中并不出现,也不影响程序的执行。注释内容以分号“;”开始,可以为任何字符,注释内容占多行时,每行都必须以分号开始。

每一条汇编指令都必须有操作码,一条语句必须在一行之内写完。

### 4.1.2 汇编伪指令

汇编语言除了定义汇编执行指令外,还定义了一些汇编伪指令,伪指令也称为汇编程序控制译码指令,属于说明性汇编指令。伪指令提供汇编时的某些控制信息,用来对汇编过程进行控制和操作。伪指令汇编时不产生机器语言代码,是 CPU 不能执行的指令,不影响程序的执行。

不同汇编程序伪指令的规定略有不同,常用的伪指令如下。

#### 1. 定位伪指令,ORG(Origin)

格式: ORG 操作数

此伪指令的操作数常为一个 16 位的二进制数,它指出了该指令后的指令的第一个字节在程序存储器中的地址,即生成目标代码或数据块的起始存储地址。必须放在每段源程序或数据段的开始行。在一个源程序中,可以多次定义 ORG 伪指令,但每次定义不应

和前面生成的机器指令的存放地址重叠。

```

例 4-1      ORG      0200H
              START:  MOV      A, #80H
              MOV      R1, A
              ⋮
              ORG      0500H
              NEXT:   MOV      DPTR, #7FFFH
              MOV      A, @DPTR
              ⋮
  
```

以 START 开始的程序编译为目标代码后,从 0200H 开始连续存放;以 NEXT 开始的程序目标代码则从 0500H 存储单元开始连续存放。注意从 START 开始的程序段所占用的程序地址最多到 04FFH,否则会与从 NEXT 开始的程序段地址发生重叠。重叠程序编译时不会产生错误,但运行时肯定会发生错误,所以在设置程序段的开始地址时要保证各程序段地址不重叠。

## 2. 结束汇编伪指令,END

格式: END

该伪指令必须安排在汇编源程序的末尾。在一个程序中,只允许出现一条 END 伪指令,汇编程序遇到 END 伪指令就结束,对 END 伪指令后面的所有语句都不进行编译。

## 3. 定义字节伪指令,DB(Define Byte)

格式: [标号:] DB  $X_1, X_2, X_3, \dots, X_n$

该伪指令将其右边的数据依次存放以左边标号为起始地址的存储单元中。 $X_i$  为单字节数据,可以采用二进制、十进制、十六进制和 ASCII 码等多种形式。标号可有可无。

```

例 4-2      ORG      1000H
              TAB:   DB      3FH, 06H, 25
              DB      'MCS-51'
              ⋮
  
```

经汇编后,地址 1000H 开始的存储单元的内容为:

```

(1000H) = 3FH
(1001H) = 06H
(1002H) = 19H
(1003H) = 4DH      ;M 的 ASCII 码
(1004H) = 43H
(1005H) = 53H
(1006H) = 2DH
(1007H) = 35H
(1008H) = 31H
  
```

单引号表示其中内容为字符,目标代码用 ASCII 码表示。DB 指令常用在查表程序中。



#### 4. 定义双字节数据伪指令, DW(Define Word)

格式: [标号:] DW  $Y_1, Y_2, Y_3, \dots, Y_n$

该伪指令与 DB 伪指令的不同之处是, DW 定义的是双字节数据而 DB 定义的是单字节数据, 其他用法相同。存放时按照高位字节在前、低位字节在后的原则, 即每个双字节的高 8 位数据在低地址单元, 低 8 位数据在高地址单元, 主要用于定义 16 位地址。

**例 4-3**

```

ORG      8000H
TAB:     DW      1234H, 9AH, 10
END

```

汇编后存储单元内容为:

```

(8000H) = 12H      (8001H) = 34H
(8002H) = 00H      (8003H) = 9AH
(8004H) = 00H      (8005H) = 0AH

```

注意该伪指令中数据为单字节时, 高 8 位补零。

#### 5. 赋值伪指令, EQU(Equal)

格式: 字符名称 EQU 项(数或汇编符号)

该伪指令将“项”赋给“字符名称”。字符名称不等于标号(注意字符名称后没有冒号), “项”可以是数(8 位或者 16 位), 也可以是汇编符号。用 EQU 赋值的符号名可以用做数据地址、代码地址、位地址或一个立即数。“字符名称”必须先赋值后使用, 通常将赋值语句放在源程序的开头。

**例 4-4**

```

ORG      1000H
AA       EQU      R1
A20      EQU      20H
DELAY    EQU      1567H
MOV      R0, A20   ; (R0) ← (20H)
MOV      A, AA     ; (A) ← (R1)
LCALL    DELAY     ; 调用起始地址为 1567H 的子程序

```

EQU 赋值后, AA 为寄存器 R1 的地址, A20 为 RAM 直接地址 20H, DELAY 为 16 位地址 1567H。

#### 6. 数据地址赋值伪指令, DATA

格式: 字符名称 DATA 表达式

该伪指令将右边“表达式”的值赋给左边的“字符名称”。表达式可以是一个 8 位或 16 位的数据或地址, 也可以是包含所定义“字符名称”在内的表达式, 但不能是汇编符号(如 R0 等)。有些汇编程序只允许 DATA 定义 8 位的数据或地址, 16 位地址需用 XDATA 伪指令定义, XDATA 定义的伪指令格式与 DATA 伪指令格式相同。

DATA 伪指令定义的“字符名称”不同于 EQU 定义的“字符名称”, 没有先定义后使用的限制, DATA 伪指令通常放在程序的开头或末尾。

#### 7. 位地址赋值伪指令, BIT

格式: 字符名称 BIT 位地址

该伪指令将位地址赋给“字符名称”，只能用于可以进行位操作的位地址单元，常用于有位操作的程序中。

**例 4-5**    P10            BIT        90H  
             FLAG2        BIT        02H

位地址 90H(P1 口 D0 位)赋给 P10,FLAG2 的值为 02H 位中的数值。在以后程序编写时用 P10 直接代替 90H 位,FLAG2 直接代替 02H 位(RAM 的 20H 单元中的 D2 位),提高程序的可读性。

#### 8. 定义存储空间伪指令,DS(Define Storage)

格式:    DS        表达式

该伪指令是指汇编时,从指定的地址单元开始(如由标号或 ORG 指令指定首址),保留由表达式设定的若干存储单元作为备用空间。

**例 4-6**    ORG        2000H  
             DS        07H  
             DB        20H,20  
             DW        12H

汇编后,从地址 2000H(包含 2000H)开始保留 7 个存储单元,2007H 开始存储内容依次为:

(2007H) = 20H        (2008H) = 14H  
(2009H) = 00H        (200AH) = 12H

注意,DB、DW、DS 伪指令都只对程序存储器起作用,不能对数据存储器进行初始化。

## 4.2 寻址方式

根据指令格式,要正确执行指令必须要得到正确的操作数。所谓寻址就是指寻找指令中操作数所在的地址。地址泛指一个存储单元或某个寄存器。寻址方式越多样、越灵活,指令系统将越高效,计算机的功能也随之越强。用高级语言编程时,因为计算机内存空间大,硬盘容量大,程序员不必过多关心程序和数据的内存空间安排问题,例如,C 语言编写的以下程序:

      x = 20;            y = 30;  
      z = x + y;

程序员只要知道数据 20 存放在代号为  $x$  的单元,30 存放在代号为  $y$  的单元,相加的结果存放在代号为  $z$  的单元中,至于  $x, y, z$  在内存中具体的存放地址则根本不必关心。但汇编语言不同于高级语言,程序设计时要针对系统的硬件环境编程,数据的存放、传送、运算都要通过指令来完成,程序员必须由始至终都十分清楚操作数的位置、RAM 空间的占用情况等,以便将它们传送至适当的空间并有足够的空间去操作。因此,汇编编程时如何寻找存放操作数的空间位置和提取操作数就显得十分重要了。所谓寻址方式就是如何找到存放操作数的地址并把操作数提取出来的方法。它是单片机的重要性能指标之一,也是汇编语言程序设计中最基本的内容之一,关系到程序是否能正常执行,必须牢固掌握。

MCS-51 单片机指令系统的寻址方式有 7 种,寄存器寻址、直接寻址、立即寻址、寄存器间接寻址、变址间接寻址、相对寻址和位寻址。寻址方式通常是针对源操作数,否则需特别指明是针对目的操作数,以免出错。不管什么样的寻址方式,目的都是正确取出参与操作的操作数,掌握寻址方法可以帮助程序的编写和调试。

### 1. 寄存器寻址

寄存器寻址方式以指令中给出的某一寄存器的内容作为操作数。可以实现寄存器寻址操作的寄存器包括寄存器组 R0~R7,累加器 ACC,寄存器 B,数据指针 DPTR 和进位 C 等。

**例 4-7**    MOV        A,        R0     ; (A)←(R0)  
             MOV        P1,        A     ; (P1)←(A)  
             INC        R0               ; (R0) + 1→(R0)

该程序段首先将 R0 中的内容送累加器 A,经 P1 口输出后,R0 中的内容加 1。R0 代表 RAM 中的地址,如果寄存器组为 0 区,则将(00H)单元的内容送累加器 A(0E0H),A 中内容送 P1 口,(00H)中内容加 1。该寻址方式中,源操作数为寄存器的内容。

### 2. 直接寻址

直接寻址方式在指令中直接给出操作数所在存储单元的地址,该地址指出了参与运算或传送的数据所在的字节单元或位的地址。

直接寻址方式中操作数存储的空间有以下 3 种:

- 特殊功能寄存器 SFR;
- 片内 RAM 的低 128 B(00H~7FH);
- 位地址空间。

对于特殊功能寄存器直接寻址时,可使用它们的地址,也可使用它们的寄存器名。

**例 4-8**        MOV        A,        30H     ; (30H)→(A)

片内 RAM 中 30H 单元中的内容送累加器 A。

             MOV        A,        P1       ; (P1)→(A)

SFR 中 P1 口内容送累加器 A,也可写成:

             MOV        A,        90H     ; (90H)→(A)

其中 90H 为 P1 口的地址。

访问特殊功能寄存器只能用直接寻址方式。因 89S52 内部 RAM 空间为 256 B,高 128 B 与特殊功能寄存器被并列设置在 80H~FFH 地址空间的物理层,这意味着它们的地址相同但物理空间不同。访问高 128 B 地址空间时,使用寄存器间接寻址。如:

MOV        0A0H,    #20H     ; #20H→(P2)  
MOV        A,        0A0H     ; 直接寻址,(P2)→(A)  
MOV        R0,       #0A0H     ;  
MOV        @R0,     #40H     ; #40H→(0A0)  
MOV        A,        @R0     ; 寄存器间接寻址,访问高 128 B RAM

执行结果为,P2 口(0A0H):20H;内部 RAM(0A0H):40H。应注意其区别。

### 3. 立即寻址

立即寻址方式在指令中直接给出参与操作的常数,操作码后面紧跟 1 B 或 2 B 的操作数(称为立即数)。立即寻址时操作数以指令的形式存放于程序存储器中,不占用内部 RAM 单元。指令中的操作数前面必须以 # 号标识,可以是一个 8 位或 16 位的二进制常数,也可以用十进制或十六进制表示。

**例 4-9**      MOV        A,     30H        ; (30H)→(A)  
                 MOV        A,     #30H       ; #30H→(A)

第一条程序将 RAM 中 30H 单元的内容送累加器 A,程序执行后 A 中的内容由 30H 单元决定,采用直接寻址的方法;第二条程序把立即数 30H 送累加器 A,程序执行后 A 中的内容为 30H,采用立即寻址的方法。

**例 4-10**   MOV   DPTR,    #0EFFFH       ; 0EFFFH→(DPTR)

将 16 位立即数 EFFFH 送入 16 位数据指针 DPTR,该条程序执行后,DPTR = EFFFH。

### 4. 寄存器间接寻址

寄存器间接寻址方式以指令中寄存器的内容作为地址,而该地址单元的内容为操作数。这是一种二次寻址方式,所以称为寄存器间接寻址。

程序执行分两步完成:首先根据指令得到寄存器的内容,即操作数的地址;然后根据地址找到所需要的操作数,并完成相应的操作。

在寄存器间接寻址指令中,采用 R0、R1 或 DPTR 作为地址指针,即存放地址的寄存器,加 @ 号标识。

**例 4-11**      MOV        40H,    #36H        ; 36H→(40H)  
                 MOV        R0,     #40H        ; 40H→(R0)  
                 MOV        A,     @R0         ; ((R0))→(A)

该程序段首先将立即数 36H 送内部 RAM 的 40H 单元,R0 中送立即数 40H,则 R0 的内容为 40H,执行间接寻址指令后,累加器 A 中内容为立即数 36H。指令执行时先查出 R0 中的内容为 40H,再根据地址 40H 取出其中的操作数送累加器 A。间接寻址过程如图 4.2.1 所示。

利用地址指针进行寄存器间接寻址可以拓宽单片机的寻址范围,使其既可以访问内部 RAM 的 256 B,也可以访问外部 RAM 的 64 KB 空间。其中:

@Ri 用于片内 RAM 寻址时,地址范围为 00H~FFH,如“MOV A,@R0”。

@Ri 用于片外 RAM 寻址时,寻址空间为 00H~FFH;此时片外 RAM 地址的低 8 位由 Ri 中的内容决定,地址高 8 位一般由 P2 口决定,如“MOVX A,@R0”。应注意 MOV 与 MOVX 的区别,MOV 用于内部 RAM 操作,MOVX 用于外部 RAM 操作。

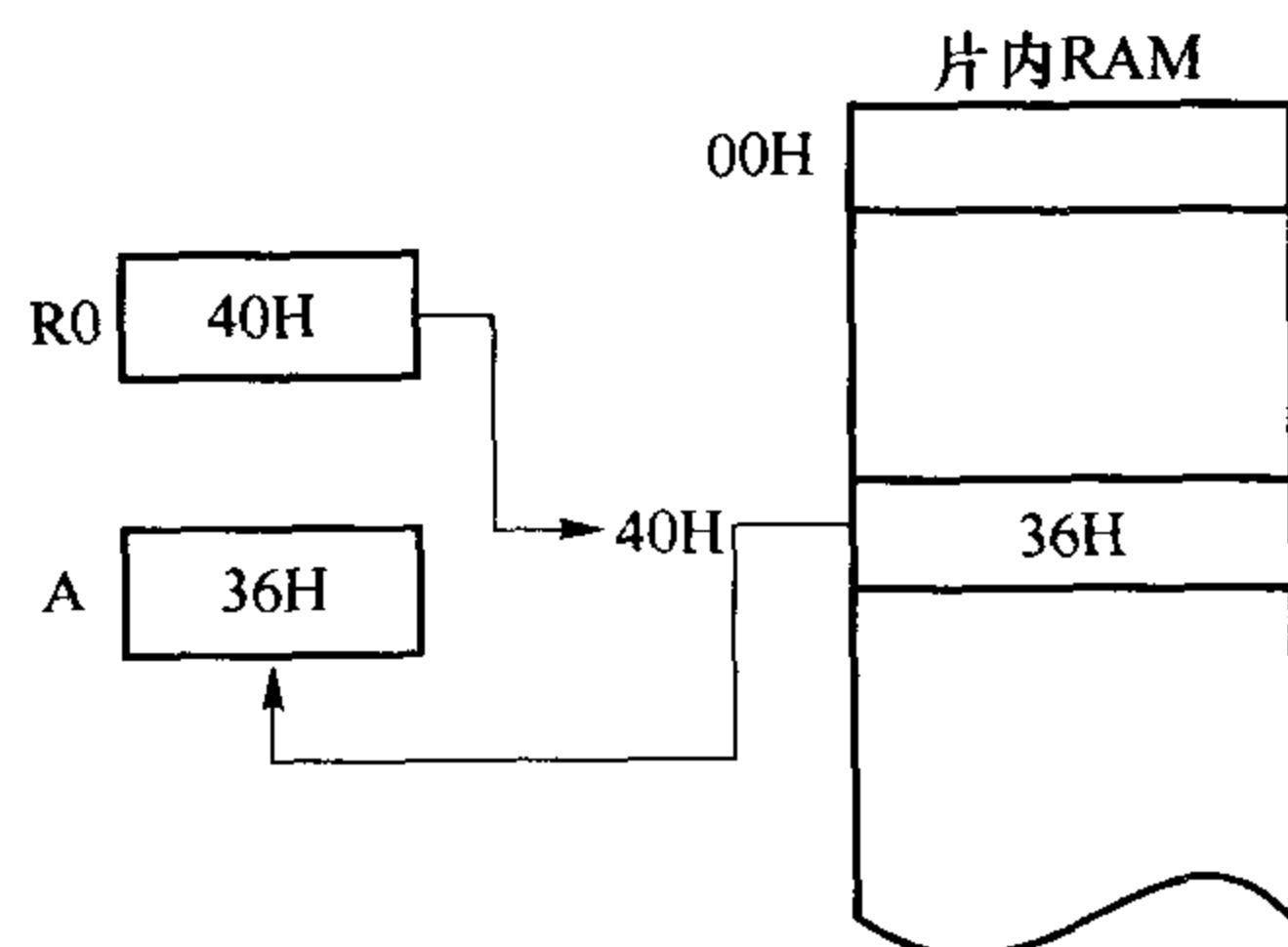


图 4.2.1 MOV A,@R0 指令执行过程

**例 4-12**    `MOV     R0,     # 06H            ; 06H→(R0)`  
              `MOVX    A,        @R0            ; ((R0))→(A)`

该程序将外部 RAM 中地址  $\times\times 06H$  单元的内容送累加器 A,  $\times\times$  代表外部 RAM 的高 8 位地址, 一般由 P2 口决定。

@DPTR 的寻址范围覆盖片外 RAM 的全部 64 KB 区域, 如“`MOVX A, @DPTR`”, 注意指令中采用 MOVX。

**例 4-13**    `MOV           DPTR,   # 0EFFFH       ; 0EFFFH→(DPTR)`  
              `MOVX          A,        @DPTR        ; (DPTR)→(A)`

该程序将外部 RAM 中地址 0EFFFH 单元的内容送累加器 A。

### 5. 变址间接寻址 (基址寄存器 + 变址寄存器间接寻址)

变址间接寻址指令由基址寄存器和变址寄存器组成, 16 位寄存器 DPTR(数据指针)或 PC(程序计数器)作为基址寄存器, 8 位累加器 A 作为变址寄存器。基址寄存器内容和变址寄存器内容相加形成新的 16 位地址, 该地址即为操作数的存储地址。这是一种独特的寻址方式, A 中的内容可以随程序的运行动态变化, 所以可以实现动态寻址。

变址寻址方式只能访问程序存储器, 访问时只能从程序存储器中读入数据, 而不能写出数据, 所以这种变址寻址方式多用于查表操作。

指令系统中有如下两条单字节、双周期的变址寻址指令:

`MOVC     A,        @A + PC            ; ((A) + (PC))→(A)`  
`MOVC     A,        @A + DPTR        ; ((A) + (DPTR))→(A)`

**例 4-14**    查共阴极数码管对应的显示代码程序。

```

MOV        A,        30H            ; (30H)→(A)
MOV        DPTR,    # SEGTAB       ; 8000H→(DPTR)
MOVC       A,        @A + DPTR     ; ((A) + (DPTR))→(A)
:
ORG        8000H
SEGTAB:    DB        3FH, 06H, 5BH, 4FH, 66H   ; 对应于字符 0、1、2、3、4
           DB        6DH, 7DH, 07H, 7FH, 67H   ; 对应于字符 5、6、7、8、9

```

其中, 30H 中的内容为要显示数据 0~9。当要显示字符 0 时, 30H 地址中内容为 0, 累加器 A 中内容为 0, 则执行 `MOVC A, @A+DPTR` 指令后, 将 8000H 中内容送累加器 A, A 中内容为 3FH; 当要显示字符 8 时, 30H 地址中内容为 8, 累加器 A 中内容为 8, 则执行 `MOVC A, @A+DPTR` 指令后, 将 8008H 中内容送累加器 A, A 中内容为 7FH; 依此类推。

### 6. 相对寻址

相对寻址以当前程序计数器 PC 值为基准, 加上指令中给定的偏移量 rel 所得结果而形成实际的转移地址。这种寻址方式主要用于转移指令到指定转移的目标地址。一般将相对转移指令操作码所在地址称为源地址, 转移后的地址称为目的地址, 目的地址的计算方法如下:

$$\text{目的地址} = \text{源地址} + \text{相对转移指令字节数}(2 \text{ 或 } 3) + \text{rel}$$

相对转移指令字节数为 2 或 3,这是因为 AT89S52 指令系统中既有双字节转移指令,也有三字节转移指令,双字节指令加 2,三字节指令则加 3。

偏移量 rel 为有符号数,取值范围为  $-128 \sim +127$ ,程序中一般以 8 位二进制补码形式表示。

#### 例 4-15

```
ORG    1000H
JC      75H           ;rel = 75H
```

这是一条以 Cy 为条件的双字节转移指令。指令的功能是检测进位 Cy,当  $Cy=1$  时,程序转向  $(PC)+2+(rel)$  所指示的目的地址开始执行;当  $Cy=0$  时,程序继续往下执行。

设  $Cy=1$ ,则转移后的目的地址为:  $1000H+2+75H=1077H$ ,程序转到 1077H 单元执行。

#### 7. 位寻址

AT89S52 单片机具有很强的位处理能力。操作数不仅可以按字节为单位进行存取和操作,而且也可以按 8 位二进制数中的某一位为单位进行存取和操作,此时的操作数地址就称为位地址。位寻址方式指操作数是 8 位二进制中的某一位(即位地址),指令中位地址用 bit 表示。

AT89S52 片内 RAM 有两个区域可以进行位寻址:其一是 20H~2FH 的 16 个单元共 128 位的位地址;其二是字节地址为 8 的倍数的 12 个特殊功能寄存器,共 92 个位地址。

位地址常用以下 4 种方式表示:

- 直接使用位寻址空间中的位地址,如 7FH;
- 采用第几字节单元第几位的表示方法,如上述位地址 7FH 可以表示成 2FH. 7;
- 对于特殊功能寄存器,可以直接用寄存器名字加位数的方法,如累加器中最低位 D0 可以表示成 ACC. 0;
- 经伪指令定义过的字符名称,详见 4.1.2 小节。

#### 例 4-16

```
MOV     C,    7FH           ;(7FH)→Cy
MOV     C,    2FH. 7        ;(7FH)→ Cy
MOV     C,    ACC. 0        ;(ACC. 0)→ Cy
```

虽然 AT89S52 单片机的寻址方式有多种,但指令对哪一个存储器空间进行操作是由指令的操作码和寻址方式确定的。总的来说,有以下几个原则:

- 对程序存储器只能采用立即寻址和变址寻址方式;
- 对特殊功能寄存器空间只能采用直接寻址方式,不能采用寄存器间接寻址方式;
- 内部数据存储器高 128 B 只能采用寄存器间接寻址方式,不能采用直接寻址方式;
- 内部数据存储器低 128 B 既能采用寄存器间接寻址方式,又能采用直接寻址方式;
- 外部扩展数据存储器只能采用 MOVX 指令访问。

**例 4-17** 判断下列指令源操作数和目的操作数各自的寻址方式。

(1) MOV A,	#65H
寄存器寻址	立即数寻址
(2) MOV @R1,	65H
寄存器间接寻址	直接寻址
(3) MOV 30H,	R2
直接寻址	寄存器寻址
(4) MOV C,	20H
位寻址	位寻址
(5) DJNZ R2,	LOOP
寄存器寻址	相对寻址
(6) MOV 60H,	@R1
直接寻址	寄存器间接寻址
(7) MOVC A,	@A + DPTR
寄存器寻址	变址寻址

### 4.3 指令系统

AT89S52 单片机指令系统具有 111 条指令,按指令执行时间进行分类,有 64 条单周期指令,45 条双周期指令和 2 条(乘法、除法指令)四周期指令。若取振荡频率为 12 MHz,则 AT89S 系列单片机大多数指令的执行时间仅需  $1\mu\text{s}$ (即一个机器周期),所以该指令系统具有简单易学、存储效率高、执行速度快等特点。

AT89S52 单片机的指令系统按其功能可分为:

- 数据传送指令 29 条;
- 算术运算指令 24 条;
- 逻辑运算指令 24 条;
- 控制转移指令 17 条;
- 位操作指令(布尔操作)17 条。

#### 4.3.1 数据传送指令

CPU 在进行算术和逻辑操作时,绝大多数指令都有操作数,所以数据传送是一种最基本、最主要的操作。在通常的应用程序中,传送指令占有极大的比例,数据传送是否灵活、迅速,对整个程序的编写和执行都起着很大的作用。

所谓“传送”,是把源地址单元的内容传送到目的地址单元中去,指令执行后一般源地址单元内容不变;或者源地址单元与目的地址单元内容互换。数据传送类指令分为 3 类:数据传送、数据交换和栈操作。

AT89S52 提供了极其丰富的数据传送指令,其数据传送指令操作可以在累加器 A、工作寄存器 R0~R7、内部数据存储器、外部数据存储器和程序存储器之间进行,如图 4.3.1 所示。这类指令共有 29 条,数据传送指令助记符为 MOV,MOVB,MOVC;数据交



换指令助记符 XCH, XCHD, SWAP; 栈指令助记符 PUSH, POP。

执行数据传送类指令时, 除以累加器 A 为目的操作数的指令会对奇偶标志位 P 有影响外, 其余指令执行时均不会影响任何标志位。

注: 累加器在指令中一般用 A 表示, 但在堆栈、位寻址类指令中须用 ACC 表示, 如:

PUSH ACC

sefB ACC.6 2

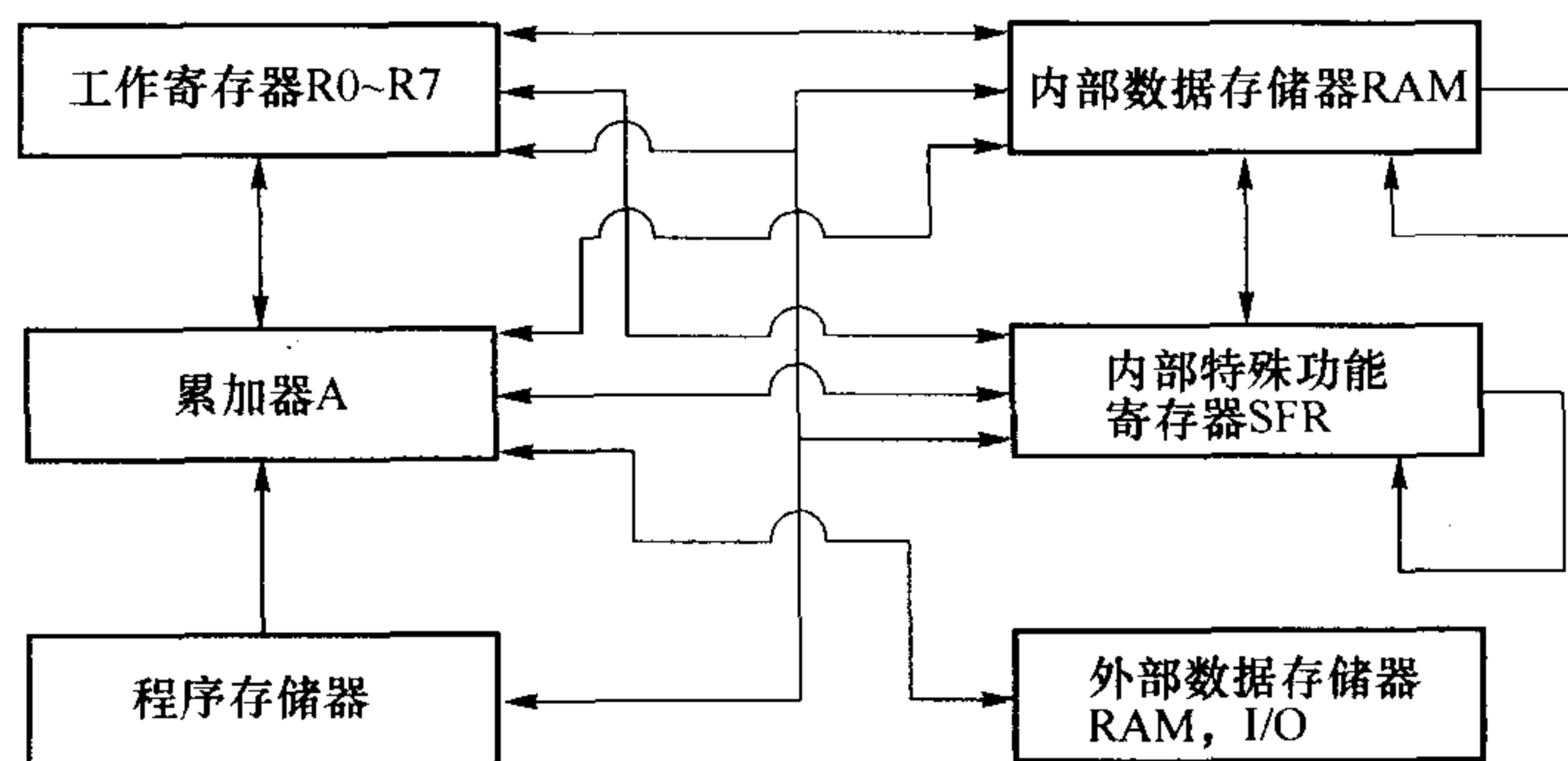


图 4.3.1 数据传送操作

### 1. 以累加器 A 为目的操作数的指令, MOV (Move)

MOV A, Rn ; 寄存器寻址, (Rn) → (A), n = 0 ~ 7

MOV A, direct ; 直接寻址, (direct) → (A)

MOV A, @Ri ; 寄存器间接寻址, ((Ri)) → (A), i = 0 或 1

MOV A, #data ; 立即寻址, data → (A)

注: 以下指令中, 工作寄存器 R 的标记为 n 时, n = 0 ~ 7; 当标记为 i 时, i = 0 或 1。

这组指令的功能是把源操作数的内容送累加器 A, 源操作数内容不变。

例 4-18 分析程序执行后寄存器的内容。

MOV A, R6 ; 若 (R6) = 50H, 则执行后 (A) = 50H, (R6) = 50H 不变

MOV A, 30H ; 若 (30H) = 20H, 则执行后 (A) = 20H, (30H) = 20H 不变

MOV A, @R0 ; 若 (R0) = 30H, (30H) = 50H, 则 (A) = 50H, (R0) = 30H

MOV A, #18H ; 执行后 (A) = 18H, P = 0

### 2. 以寄存器 Rn 为目的操作数的指令

MOV Rn, A ; 寄存器寻址, (A) → (Rn)

MOV Rn, direct ; 直接寻址, (direct) → (Rn)

MOV Rn, #data ; 立即寻址, data → (Rn)

这组指令的功能是把源操作数的内容送当前工作寄存器组 R0 ~ R7 中的某个寄存器, 源操作数的内容不变。

例 4-19 若 (A) = 30H, (30H) = 58H, 分析程序执行后寄存器的内容。

MOV R2, A ; (R2) = 30H

MOV R2, 30H ; (R2) = 58H

MOV R2, #30H ; (R2) = 30H

### 3. 以直接地址为目的操作数的指令

MOV direct, A ; 寄存器寻址, (A) → (direct)

```

MOV    direct,   Rn        ;寄存器寻址,(Rn)→(direct)
MOV    direct1,  direct2    ;直接寻址,(direct2)→(direct1)
MOV    direct,   @Ri        ;寄存器间接寻址,((Ri))→(direct)
MOV    direct,   #data      ;立即寻址,data→(direct)

```

这组指令的功能是把源操作数的内容送直接地址单元,源操作数的内容不变。

**例 4-20** 若(A)=30H,(R2)=58H,(40H)=68H,(R0)=40H,分析程序执行后寄存器的内容。

```

MOV    P1,   A            ;(P1)=30H,(A)=30H
MOV    30H,  R2           ;(30H)=58H,(R2)=58H
MOV    31H,  40H          ;(31H)=68H,(40H)=68H
MOV    32H,  @R0          ;(32H)=68H,(R0)=40H
MOV    33H,  #45H         ;(33H)=45H

```

#### 4. 以间接地址为目的操作数的指令

```

MOV    @Ri, A            ;寄存器寻址,(A)→((Ri))
MOV    @Ri, direct       ;直接寻址,(direct)→((Ri))
MOV    @Ri, #data        ;立即寻址,data→((Ri))

```

这组指令的功能是把源操作数的内容送到以 R0 或 R1 的内容作为地址的内部 RAM 单元,源操作数的内容不变。

**例 4-21** 若(A)=30H,(32H)=58H,(40H)=68H,(R0)=40H,分析程序执行后寄存器的内容。

```

MOV    @R0, A            ;(40H)=30H,(A)=30H
MOV    @R0, 32H          ;(40H)=58H,(32H)=58H
MOV    @R0, #76H         ;(40H)=76H

```

#### 5. 16 位数据传送指令

```

MOV    DPTR, #data16     ;立即寻址,data16→(DPTR)
                        ;即 dataH→(DPH),dataL→(DPL)

```

这条指令的功能是把 16 位立即数传送到数据指针 DPTR,16 位数据的高 8 位传送到 DPH,低 8 位传送到 DPL。AT89S52 设有两个 DPTR,通过 AUXR1 中 DPS 位的数值进行选择,当 DPS 位设置为 1 时,则指令中的 DPTR 即为 DPTR1,DPTR0 被屏蔽,DPS 位设置为 0 时,则指令中的 DPTR 即为 DPTR0,DPTR1 被屏蔽。设有两个 DPTR,如寄存器区一样,可避免频繁的出入栈操作。AUXR1 特殊功能寄存器的格式如图 4.3.2 所示。

位	D7	D6	D5	D4	D3	D2	D1	D0	字节地址: A2H
	—	—	—	—	—	—	—	DPS	不能位寻址

图 4.3.2 AUXR1 特殊功能寄存器的格式

复位后  $DPS=0$ , 自动选择  $DPTR0$ 。

**例 4-22** 分析程序执行后寄存器  $DPH$  和  $DPL$  的内容。

```
MOV    A2H,    #01H    ;选中 DPTR1
MOV    DPTR,    #5678H    ;(DP1H) = 56H, (DP1L) = 78H, (DPTR1) = 5678H
```

## 6. 查表指令, MOVC(Move Code)

```
MOVC    A,    @A + PC    ;变址寻址, ((A) + (PC)) → (A)
MOVC    A,    @A + DPTR ;变址寻址, ((A) + (DPTR)) → (A)
```

这组指令的功能是将基址寄存器( $PC$  或  $DPTR$ )的内容与变址寄存器  $A$  的内容相加, 组成新的 16 位地址, 该新地址单元的内容送累加器  $A$ 。

这两条指令专门用于访问程序存储器中的数据表。注意  $PC$  指下一条指令的地址。由于  $A$  的内容在  $0\sim 255(FFH)$  之间, 所以  $(A) + (PC)$  所得到的新地址只能在该查表指令以后 256 B 单元内, 表格的大小受到了限制, 称为近程查表。 $(A) + (DPTR)$  可在 64 KB 程序存储器内任意安排, 称为远程查表。

**例 4-23** 共阴极数码管对应的显示代码程序。

```
ORG      8000H
MOV      A,      30H          ;PC = 8000, 双字节指令
ADD      A,      #80H         ;PC = 8002, 双字节指令
MOVC     A,      @A + PC      ;PC = 8004, 双字节指令
                                ;PC = 8006
:
ORG      8086H
SEG TAB: DB      3FH, 06H, 5BH, 4FH, 66H ;对应于字符 0,1,2,3,4
          DB      6DH, 7DH, 07H, 7FH, 67H ;对应于字符 5,6,7,8,9
```

其中,  $30H$  中内容为要显示的数据  $0\sim 9$ 。当要显示字符 0 时,  $30H$  中内容为 0, 累加器  $A$  中内容为  $80H+0=80H$ , 则执行  $MOVC\ A, @A+PC$  指令后, 将  $8086H$  中内容送累加器  $A$ , 则  $A$  中内容为  $3FH$ ; 当要显示字符 8 时,  $30H$  内容为 8, 累加器  $A$  中内容为  $88H$ , 则执行  $MOVC\ A, @A+PC$  指令后, 将  $808EH$  中内容送累加器  $A$ , 则  $A$  中内容为  $7FH$ 。如此可以取出对应数字的显示码。

## 7. 累加器 A 与片外 RAM 数据传送指令, MOVX(Move External Ram)

```
MOVX    A,      @Ri          ;寄存器间接寻址, ((Ri)) → (A), 且使  $\overline{RD}=0$ 
MOVX    A,      @DPTR        ;寄存器间接寻址, ((DPTR)) → (A), 且使  $\overline{RD}=0$ 
MOVX    @Ri,    A            ;寄存器寻址, (A) → ((Ri)), 且使  $\overline{WR}=0$ 
MOVX    @DPTR,  A            ;寄存器寻址, (A) → ((DPTR)), 且使  $\overline{WR}=0$ 
```

这组指令的功能是实现累加器  $A$  中内容与外部扩展的 256 B、64 KB RAM、I/O 之间的数据传送。读写线的状态自动改变。前两条为单片机数据读入指令, 所以读信号为低电平, 写信号为高电平; 后两条为单片机写出指令, 写信号为低电平, 而读信号为高电平。

**例 4-24** 编写将外部 RAM 的  $0088H$  单元中存放的数据传送到外部 RAM 的  $1818H$  单元的程序。

**解** 外部 RAM 的 0088H 单元中的数据不能直接传送到外部 RAM 的 1818H 单元, 必须经过累加器 A 的转传。首先利用累加器 A 取出 0088H 单元的内容, 再将 A 中的内容传送到 1818H 单元。相应程序为:

```
MOV    A2H,    #01H    ;选中 DPTR1
MOV    P2,     #00H    ;立即数 00H→(P2)
MOV    R0,     #88H    ;立即数 88H→(R0)
MOV    DPTR,   #1818H  ;立即数 1818H→(DPTR1)
MOVSX  A,      @R0     ;(0088H)→(A)
MOVSX  @DPTR,  A       ;(A)→(1818H)
```

该段程序也可以改写为:

```
MOV    A2H,    #01H    ;选中 DPTR1
MOV    DPTR,   #0088H  ;0088H→(DPTR1)
MOVSX  A,      @DPTR   ;(0088H)→(A)
MOV    DPTR,   #1818H  ;1818H→(DPTR1)
MOVSX  @DPTR,  A       ;(A)→(1818H)
```

## 8. 堆栈操作指令, PUSH, POP

```
PUSH    direct          ;直接寻址, (SP) + 1→(SP), (direct)→((SP))
POP     direct          ;直接寻址, ((SP))→(direct), (SP) - 1→(SP)
```

堆栈指令是根据堆栈指针 SP 中栈顶地址进行数据传送的。

第一条为压栈指令, 首先将堆栈栈顶指针 SP 内容加 1, 然后把 direct 地址中的内容传送到堆栈指针 SP 指示的内部 RAM 单元。

第二条为弹栈指令, 用于把堆栈指针 SP 指示的内部 RAM 单元中的内容传送到 direct 单元, 然后堆栈指针 SP 减 1, 指向新的栈顶地址。

堆栈为空的标志是栈顶地址和栈底地址重合。堆栈操作遵循先入后出的原则。

**例 4-25** 设 (A) = 40H, (B) = 50H, 分析执行下列程序后, 寄存器 SP 和 DPTR 的内容。

```
MOV    SP, #60H        ;(SP) = 60H
PUSH   A                ;(SP) = 60H + 1 = 61H, (61H) = (A) = 40H
PUSH   B                ;(SP) = 61H + 1 = 62H, (62H) = (B) = 50H
POP    DPL              ;(DPL) = (62H) = 50H, (SP) = 62H - 1 = 61H
POP    DPH              ;(DPH) = (61H) = 40H, (SP) = 61H - 1 = 60H
```

程序执行完成后, (SP) = 60H, (DPTR1) = 4050H。

堆栈操作指令一般用于子程序调用、中断等数据保护或 CPU 现场保护。单片机复位后, 堆栈指针复位为 07H, 所以程序中一般需重新设置堆栈指针。

## 9. 数据交换指令, XCH(Exchange), XCHD(Exchange Low-order Digit), SWAP

```
XCH    A,      Rn       ;寄存器寻址, (A) ↔ (Rn)
XCH    A,      direct   ;直接寻址, (A) ↔ (direct)
XCH    A,      @Ri      ;寄存器间接寻址, (A) ↔ ((Ri))
```

XCHD     A,            @Ri        ; 寄存器间接寻址,  $(A)_{0\sim3} \longleftrightarrow ((Ri))_{0\sim3}$   
 SWAP     A                            ; 寄存器寻址,  $(A)_{0\sim3} \longleftrightarrow (A)_{4\sim7}$

前3条为字节交换指令,其功能是把累加器 A 中内容和源操作数的内容相互交换。

第4条为半字节交换指令,其功能是将累加器 A 中内容的低4位和源操作数内容的低4位相互交换,各自的高4位则保持不变。

第5条指令是将累加器 A 中内容的高4位与低4位交换。

**例 4-26** 已知50H 中有一个数,在0~9 范围内,编写程序得到相应的 ASCII 码。

**解** 0~9 的 ASCII 码为30H~39H,高4位为0011,低4位为相应的数据。所以利用半字节交换指令可以把0~9 转换成相应的 ASCII 码。

```
MOV     R0,      #50H      ; 50H → (R0)
MOV     A,       #30H      ; 30H → (A)
XCHD    A,       @R0       ; A 中形成相应的 ASCII 码
MOV     @R0,     A         ; ASCII 码送回 50H
```

### 4.3.2 算术运算指令

算术运算指令包括加、减、乘、除基本四则运算和加1(增量)、减1(减量)运算。除加1和减1指令外,算术运算指令影响进位 Cy、半进位 AC、溢出位 OV 三个标志位。所以在使用时要注意和利用标志位的状态变化。助记符分别为:ADD、ADDC、INC、SUBB、DEC、DA、MUL、DIV。

#### 1. 不带进位的加法指令,ADD

```
ADD     A,       Rn        ; 寄存器寻址,  $(A) + (Rn) \rightarrow (A)$ 
ADD     A,       direct    ; 直接寻址,  $(A) + (direct) \rightarrow (A)$ 
ADD     A,       @Ri       ; 寄存器间接寻址,  $(A) + ((Ri)) \rightarrow (A)$ 
ADD     A,       #data     ; 立即寻址,  $(A) + data \rightarrow (A)$ 
```

这组指令的功能是将工作寄存器、内部 RAM 单元的内容或立即数的8位无符号二进制数与累加器内容相加,所得结果存放在累加器 A 中。进位对运算结果无影响,但该组指令影响进位。

加法时单片机确定 PSW 中各标志位的规则是:

- 相加后位7有进位输出时,则 Cy 置1,否则清0;
- 相加后位3有进位输出时,则辅助进位 AC 置1,否则清0;
- 相加后如果位7有进位输出而位6没有,或者位6有进位输出而位7没有,则置位溢出标志 OV,否则清0;
- OV=1,表示两个正数相加而和变为负数,或两个负数相加而和变为正数的错误结果;
- A 中结果里有奇数个1,则奇偶标志 P 置1,否则清0。

**例 4-27** 分析如下指令执行后累加器 A 和 PSW 中标志位的变化情况。

```
MOV     A,       #19H
ADD     A,       #66H
```

**解** 执行结果为,  $A=19H+66H=7FH$ , 用二进制表示为,  $A=01111111B$ 。

计算过程:

$$\begin{array}{r} 00011001 \\ + 01100110 \\ \hline 01111111 \end{array}$$

PSW 中各位为,  $Cy=0, AC=0, OV=0, P=1$ , 其中,  $F0, RS1, RS0$  位保持原状态不变。

这里, 如果将运算看成是两个无符号数相加, 结果是正确的; 如果将运算看成是两个带符号数相加, 结果也是正确的, 其正确性可以用  $OV=0$  来表示。

**例 4-28** 分析执行如下指令后累加器 A 和 PSW 中各标志位的变化。

```
MOV    A,      #85H
ADD    A,      #0AEH
```

**解** 执行结果为,  $85H+AEH=133H$ , 但累加器为 8 位寄存器, 所以 A 中内容取后 8 位, 则  $A=33H$ , 用二进制表示:  $A=00110011B$ 。

计算过程:

$$\begin{array}{r} 10000101 \\ + 10101110 \\ \hline (1) 00110011 \end{array}$$

PSW 中标志位为,  $Cy=1, AC=1, OV=1, P=0$ 。

如果将运算看成两个无符号数相加, 再考虑  $Cy=1$ , 结果是正确的; 如果将运算看成是两个带符号数相加, 显然结果是错的, 因为两个负数相加不可能为正, 可以由  $OV=1$  看出结果错误。

## 2. 带 Cy 进位的加法指令, ADDC (Add with Carry Flag)

```
ADDC    A,      Rn      ;寄存器寻址, (A)+(Rn)+Cy→(A)
ADDC    A,      direct  ;直接寻址, (A)+(direct)+Cy→(A)
ADDC    A,      @Ri     ;寄存器间接寻址, (A)+((Ri))+Cy→(A)
ADDC    A,      #data   ;立即寻址, (A)+data+Cy→(A)
```

这组指令的功能是将工作寄存器的内容、内部 RAM 单元的内容或立即数的 8 位无符号二进制数与进位标志一起相加, 所得结果存放在累加器 A 中。

PSW 中标志位状态的变化和不带 Cy 的加法指令相同, 主要用于多字节加法。当进位 Cy 为 0 时, 其结果与不带进位的相同。

**例 4-29**  $(A)=53H, Cy=1$ , 分析指令执行后累加器 A 和 PSW 中标志位状态的变化。

```
ADDC    A,      #0FBH
```

**解** 执行结果为:  $53H+FBH+Cy=14FH$ , 但累加器为 8 位寄存器, 所以 A 中内容取后 8 位, 则  $A=4FH$ , 用二进制表示:  $A=01001111B$ 。

计算过程:

$$\begin{array}{r} 01010011 \\ 11111011 \\ + 1 \\ \hline (1) 01001111 \end{array}$$

PSW 中标志位为,  $Cy=1, AC=0, OV=1, P=1$ 。

### 3. 加 1 指令, INC (Increment)

```

INC    A        ; 寄存器寻址,  $(A) + 1 \rightarrow (A)$ 
INC    Rn       ; 寄存器寻址,  $(Rn) + 1 \rightarrow (Rn)$ 
INC    direct   ; 直接寻址,  $(direct) + 1 \rightarrow (direct)$ 
INC    @Ri      ; 寄存器间接寻址,  $((Ri)) + 1 \rightarrow ((Ri))$ 
INC    DPTR     ; 寄存器寻址,  $(DPTR) + 1 \rightarrow (DPTR)$ 

```

加 1 指令又称为增量指令。前 4 条为 8 位数加 1 指令, 使指定变量按 8 位无符号数加 1。但只有第一条指令能影响奇偶标志位 P。若用以修改输出口(P1, P2 口等)数据时, 原来的值是从口锁存器读入而不是从引脚读入的。第 5 条指令 INC DPTR 用于对地址指针 DPTR 中内容加 1, 是 AT89S52 指令系统唯一的一条 16 位算术运算指令。

加 1 指令用于频繁修改地址指针和实现数据加 1, 通常配合寄存器间接寻址指令使用。

**例 4-30**  $(A)=0FFH, (R7)=10H, (30H)=56H$ , 分析指令执行后累加器、寄存器和 PSW 中标志位状态的变化情况。

```

MOV    A,      #0FFH
MOV    R7,     #10H
MOV    30H,    #56H
MOV    R1,     #30H
MOV    DPTR,   #8000H
INC    A
INC    R7
INC    30H
INC    @R1
INC    DPTR

```

加 1 指令依次执行结果:  $(A)=00H, P=0; (R7)=11H; (30H)=57H; (R1)=30H, (30H)=58H$ , 注意(R1)不变;  $(DPTR0)=8001H, (DP0H)=80H, (DP0L)=01H$ 。

### 4. 带 Cy 的减法指令, SUBB (Subtract With Borrow)

```

SUBB   A,  Rn      ; 寄存器寻址,  $(A) - (Rn) - Cy \rightarrow (A)$ 
SUBB   A,  direct  ; 直接寻址,  $(A) - (direct) - Cy \rightarrow (A)$ 
SUBB   A,  @Ri     ; 寄存器间接寻址,  $(A) - ((Ri)) - Cy \rightarrow (A)$ 
SUBB   A,  #data   ; 立即寻址,  $(A) - \#data - Cy \rightarrow (A)$ 

```

该组指令是把累加器 A 中的操作数减去源地址所指操作数和指令执行前的 Cy 值, 结果存放在累加器 A 中。

减法操作时单片机确定 PSW 中各标志位的规则是:

- 若减法时位 7 有借位, 则  $Cy=1$ , 否则  $Cy=0$ ;
- 若减法时位 3 有借位, 则  $AC=1$ , 否则  $AC=0$ ;



- 若减法时位 7 有借位而位 6 无借位或位 7 无借位而位 6 有借位,则  $OV=1$ , 否则  $OV=0$ ;
- $OV=1$  表示一个正数减去一个负数结果为负数, 或一个负数减去一个正数结果为正数的错误结果;
- 如果 A 中结果里有奇数个 1, 则  $P=1$ , 否则  $P=0$ 。

为了实现不带  $Cy$  的减法, 可以先将  $Cy$  清 0 (CLR C), 然后执行带  $Cy$  的减法指令。

**例 4-31** 已知  $(A)=0C9H$ ,  $(R0)=30H$ ,  $(30H)=54H$ ,  $Cy=1$ , 分析指令 SUBB A, @R0 执行后累加器和 PSW 中各标志位的变化。

**解** 执行结果:  $(A)=0C9H-54H-Cy=74H$ 。

计算过程:

$$\begin{array}{r} 11001001 \\ 01010100 \\ - \quad \quad 1 \\ \hline 01110100 \end{array}$$

PSW 中各位为:  $Cy=0$ ,  $AC=0$ ,  $OV=1$ ,  $P=0$ 。

### 5. 减 1 指令, DEC (Decrement)

- DEC A ; 寄存器寻址,  $(A)-1 \rightarrow (A)$   
 DEC Rn ; 寄存器寻址,  $(Rn)-1 \rightarrow (Rn)$   
 DEC direct ; 直接寻址,  $(direct)-1 \rightarrow (direct)$   
 DEC @Ri ; 寄存器间接寻址,  $((Ri))-1 \rightarrow ((Ri))$

这组指令使指定源操作数内容减 1。除第一条指令(对累加器 A 操作)对奇偶校验标志位 P 有影响外, 其他指令都不影响 PSW 标志位。

### 6. 十进制调整指令, DA (Decimal Adjust)

- DA A ; ①若  $AC=1$  或  $A_3 \sim 0 > 9$ , 则  $(A)+06H \rightarrow (A)$   
 ; ②若  $Cy=1$  或  $A_7 \sim 4 > 9$ , 则  $(A)+60H \rightarrow (A)$

这条专用指令常跟在 ADD 或 ADDC 指令后, 将相加后存放在累加器 A 中的结果调整为压缩的 BCD 码(Binary-Coded Decimal, 二-十进制码), 以完成十进制加法运算功能。执行该指令仅影响进位  $Cy$ 。为了保证 BCD 数相加的结果也是 BCD 数, 该指令必须紧跟在加法指令之后。

BCD 码是用二进制编码表示的十进制数, 十进制数 0~9 表示成二进制数时只需 4 位编码(0000B~1001B), 所以一个字节(8 位)可以存放两个 BCD 码, 高、低 4 位分别存放一个 BCD 码, 在一个字节中存放两个 BCD 码称为压缩 BCD 码。

注意第②步判断是在第①步判断并运算后的基础上进行的, 所以实际运行时, 由硬件对累加器 A 进行加 06H、60H 或 66H 的操作。

**例 4-32** 编制  $85H+59H$  的 BCD 码加法程序, 并分析其工作过程。

**解** 相应的 BCD 码加法程序为:

MOV A, #85H ;  $85H \rightarrow (A)$

ADD A, #59H ; 85H + 59H = 0DEH → (A)

DA A ; 44H → (A), Cy = 1

计算过程:

```

10000101
+ 01011001
-----
11011110
+   0110
-----
11100100
+ 01100000
-----
(1)01000100

```

低4位 > 9, 加06H调整

高4位 > 9, 加60H调整

执行结果:

(A) = 44H, Cy = 1, 即十进制的144D

DA A 指令执行过程如图 4.3.3 所示。

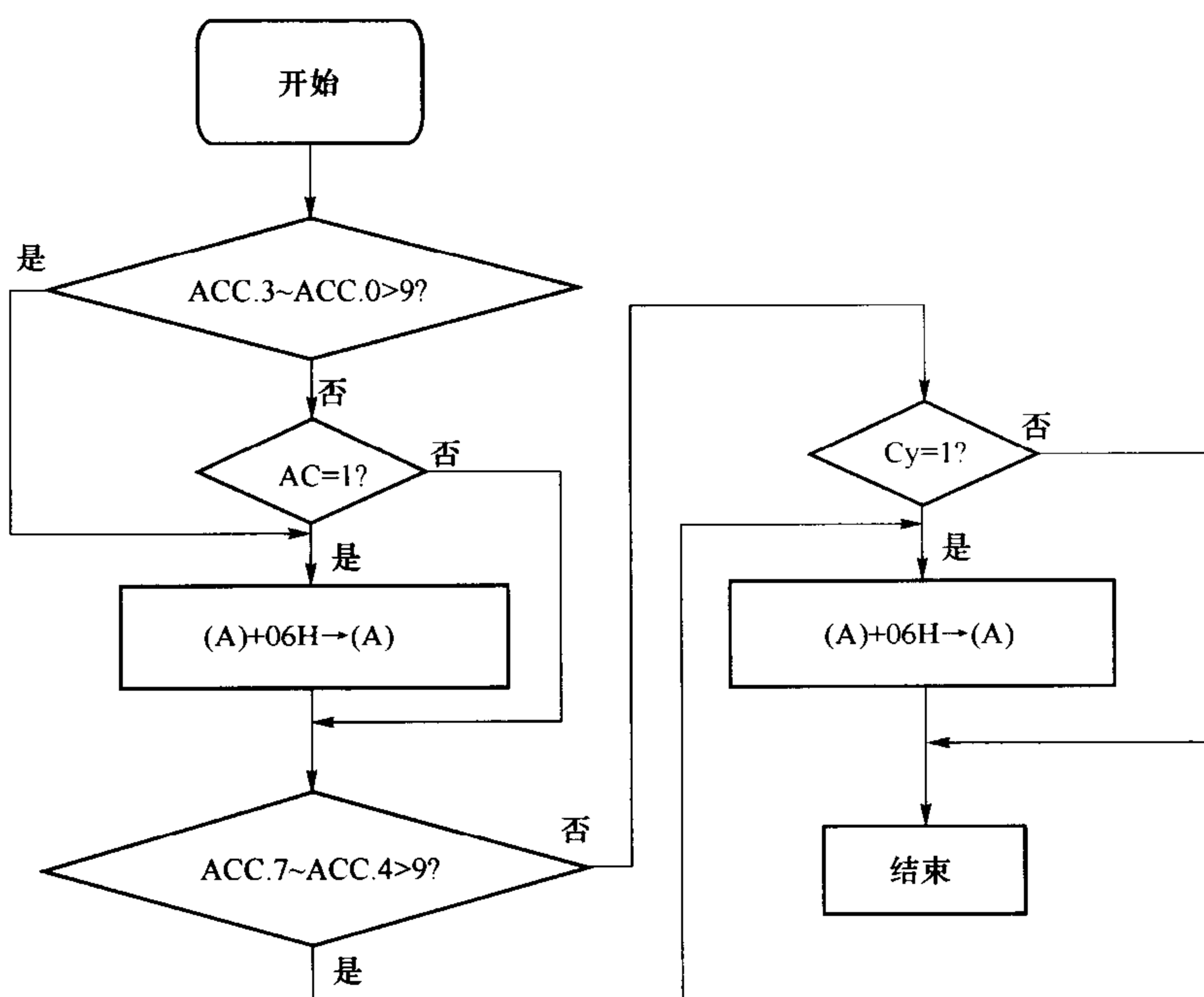


图 4.3.3 DA A 指令执行流程图

## 7. 乘法指令, MUL (Multiply)

MUL AB

乘法指令的功能是把累加器 A 和寄存器 B 中两个 8 位无符号数相乘, 并把 16 位积的低 8 位字节存于累加器 A, 高 8 位字节存于寄存器 B。如果积大于 255(0FFH), 则置位溢出标志 OV, 进位标志 Cy 总是清 0。在需要保留 Cy 值的程序中, 需先将 Cy 值转存, 待乘法指令执行完成后, 再恢复 Cy 值。

**例 4-33** 设 (A) = 50H = 80D, (B) = 0A0H = 160D, 执行指令 MUL AB, 求 A、B 的内容。

**解** 执行 MUL AB 指令后, 结果乘积为 3200H(12800D)。

(A) = 00H, (B) = 32H, OV = 1, Cy = 0

## 8. 除法指令, DIV (Division)

DIV      AB

除法指令的功能是把累加器 A 中的 8 位无符号数除以寄存器 B 中的 8 位无符号数, 所得商的整数部分保存在累加器 A 中, 余数保存在寄存器 B 中。若寄存器 B 中除数为 0, 则  $OV=1$ , 表示除法无意义; 否则  $OV=0$ 。进位标志 Cy 总是清 0。在需要保留 Cy 值的程序中, 需先将 Cy 值转存, 待除法指令执行完成后, 再恢复 Cy 值。

**例 4-34** 设  $(A)=0BFH$ ,  $(B)=32H$ , 执行指令 DIV AB, 求 A、B 的内容。

**解** 结果  $(A)=03H$ ,  $(B)=29H$ ,  $OV=0$ ,  $Cy=0$ 。

### 4.3.3 逻辑运算指令

逻辑运算指令包括清 0、求反、移位、与、或、异或等操作。操作助记符: CLR、CPL、RL、RLC、RR、RRC、ANL、ORL、XRL。

#### 1. 逻辑与指令, ANL (And Logical)

ANL      A,          Rn          ; 寄存器寻址,  $(A) \wedge (Rn) \rightarrow (A)$   
 ANL      A,          direct      ; 直接寻址,  $(A) \wedge (direct) \rightarrow (A)$   
 ANL      A,          @Ri        ; 寄存器间接寻址,  $(A) \wedge ((Ri)) \rightarrow (A)$   
 ANL      A,          #data      ; 立即寻址,  $(A) \wedge data \rightarrow (A)$   
 ANL      direct, A            ; 寄存器寻址,  $(direct) \wedge (A) \rightarrow (direct)$   
 ANL      direct, #data      ; 立即寻址,  $(direct) \wedge data \rightarrow (direct)$

前 4 条指令是将累加器 A 的内容与源地址中的操作数按位进行逻辑与操作, 结果存放在累加器 A 中。后两条指令是将直接地址单元的内容与源地址中的操作数按位进行逻辑与操作, 结果存放在直接地址单元中。

逻辑与指令可从某存储单元中取出某几位, 而把其他位变为 0。

**例 4-35** 编写程序将 RAM 中 30H 单元的压缩 BCD 码变成分离的 BCD 码, 并存放在 40H 和 41H 中。

**解**

MOV      A,          30H        ; (30H) 内容送 A  
 ANL      A,          #0F0H      ; 与 0F0H 相与, 取高 4 位  
 SWAP    A                        ; A 中内容高低 4 位交换, 变成分离 BCD 码  
 MOV      40H,      A            ; 分离 BCD 码存入 40H 单元  
 MOV      A,          30H        ; (30H) 送 A  
 ANL      A,          #0FH      ; 与 0FH 相与, 取低 4 位  
 MOV      41H,      A            ; 分离 BCD 码存入 41H

#### 2. 逻辑或指令, ORL (Or Logical)

ORL      A,          Rn          ; 寄存器寻址,  $(A) \vee (Rn) \rightarrow (A)$   
 ORL      A,          direct      ; 直接寻址,  $(A) \vee (direct) \rightarrow (A)$   
 ORL      A,          @Ri        ; 寄存器间接寻址,  $(A) \vee ((Ri)) \rightarrow (A)$

```

ORL    A,      #data    ;立即寻址, (A) ∨ data → (A)
ORL    direct, A        ;寄存器寻址, (direct) ∨ (A) → (direct)
ORL    direct, #data    ;立即寻址, (direct) ∨ data → (direct)

```

前4条指令是将累加器A的内容与源地址中的操作数按位进行逻辑或操作,结果存放在累加器A中。后两条指令是将直接地址单元的内容与源地址中的操作数按位进行逻辑或操作,结果存放在直接地址单元中。

逻辑或指令可用于使某存储单元的几位数据变为1,而其余位不变。

**例4-36** 编写程序将累加器A中低4位送入P1口低4位,而P1口高4位保持不变。

解

```

ANL    A,      #0FH      ;取A中低4位,高4位清0
ANL    P1,     #0F0H     ;使P1口低4位为0,高4位不变
ORL    P1,     A        ;累加器A低4位送入P1口低4位

```

此段程序经常用在P1口输出控制时有些位变化而有些位保持不变的情况。

### 3. 逻辑异或指令, XRL (Exclusive-Or Logical)

```

XRL    A,      Rn        ;寄存器寻址, (A) ⊕ (Rn) → (A)
XRL    A,      direct    ;直接寻址, (A) ⊕ (direct) → (A)
XRL    A,      @Ri       ;寄存器间接寻址, (A) ⊕ ((Ri)) → (A)
XRL    A,      #data     ;立即寻址, (A) ⊕ data → (direct)
XRL    direct, A        ;寄存器寻址, (direct) ⊕ (A) → (direct)
XRL    direct, #data     ;立即寻址, (direct) ⊕ data → (direct)

```

前4条指令是将累加器A的内容与源地址中的操作数按位进行逻辑异或操作,结果存放在累加器A中。后两条指令是将直接地址单元的内容与源地址中的操作数按位进行逻辑异或操作,结果存放在直接地址单元中。

逻辑异或指令可用于对某存储单元中的数据进行变换,完成其中某些位取反而其余位不变的操作。也常用于判别两操作数是否相等,若相等,结果为全0,否则不为全0。

### 4. 累加器清0和取反指令, CLR (Clear), CPL (Complement Logical)

```

CLR    A        ;寄存器寻址, 00H → (A)
CPL    A        ;寄存器寻址, (A) → (A)

```

这两条指令为单字节单周期指令,分别完成对累加器A中内容清0和逐位逻辑取反。

### 5. 移位指令, RL (Rotate Left), RLC (Rotate Left With Carry Flag), RR (Rotate right), RRC (Rotate Right With Carry Flag)

```

RL     A        ;寄存器寻址, 循环左移1位
RR     A        ;寄存器寻址, 循环右移1位
RLC    A        ;寄存器寻址, 带进位循环左移1位
RRC    A        ;寄存器寻址, 带进位循环右移1位

```

图4.3.4为循环移位指令执行示意图。注意不带进位Cy和带进位Cy的移位指令

执行时的区别。指令执行时,每次左移或右移 1 位。经常利用 RLC A 指令将累加器 A 中的内容作乘 2 运算,相对于 MUL AB 乘法指令字节数减少,指令执行速度加快。

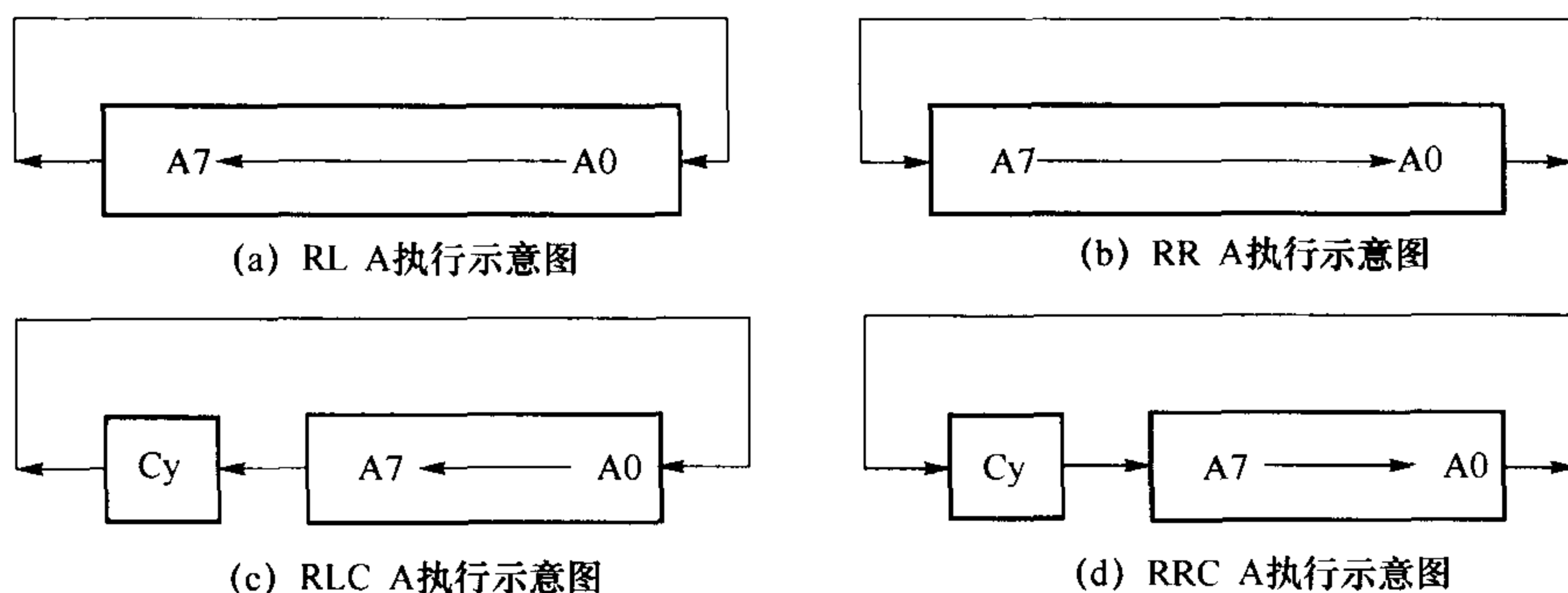


图 4.3.4 循环移位指令执行示意图

**例 4-37** 已知 16 位二进制数低 8 位存放在内部 RAM 的 M1 单元,高 8 位存放在 M1+1 单元,编写程序将其数据扩大 2 倍(设扩大后数据小于 65 536)。

**解** 利用移位指令实现。思路:利用 RLC 指令将低 8 位数据左移实现乘 2 功能,最低位需补 0,所以必须清除进位标志 Cy;再进行高 8 位数据带进位左移实现乘 2 功能。程序如下:

CLR	C		;清 Cy
MOV	R1,	#M1	;操作数低 8 位地址送 R1
MOV	A,	@R1	;操作数低 8 位数据送累加器 A
RLC	A		;低 8 位数据左移,最高位存放在 Cy 中
MOV	@R1,	A	;送回 M1 单元
INC	R1		;R1 指向 M1 + 1 单元
MOV	A,	@R1	;操作数高 8 位送 A
RLC	A		;高 8 位数据左移,M1 最高位通过 Cy 移入 ;M1 + 1 最低位
MOV	@R1,	A	;送回 M1 + 1 单元

#### 4.3.4 位(布尔)操作类指令

AT89S52 有一个布尔(BOOLEAN)处理机,它具有一套处理位变量的指令集,它以进位标志 Cy 作为位累加器 C,以 RAM 地址 20H~2FH 单元中的 128 位和地址为 8 的倍数的 SFR 的位地址单元作为操作数,进行位变量的传送、修改和逻辑操作等。有以下助记符:MOV、CLR、CPL、SETB、ANL、ORL、JC、JNC、JB、JNB、JBC 等。

##### 1. 位传送指令,MOV(Move)

MOV	C,	bit	;位寻址,(bit)←(Cy)
MOV	bit,	C	;位寻址,(Cy)→(bit)

第一条指令把由操作数指定的位变量送到 PSW 中的进位标志 Cy 中,第二条指令传送方向相反。

**例 4-38** 编程将00H位中内容和7FH位中内容互换。

**解** 位 00H 位于内部 RAM 的 20H 单元的 D0 位，7FH 位于内部 RAM 的 2FH 单元的 D7 位。设 01H 位为暂存位。图 4.3.5 为程序流程图。

```

MOV    C,    00H    ; (00H)→(Cy)
MOV    01H,  C      ; (Cy)→(01H)
MOV    C,    7FH    ; (7FH)→(Cy)
MOV    00H,  C      ; (7FH)→(00H)
MOV    C,    01H    ; (01H)→(Cy)
MOV    7FH,  C      ; (00H)→(7FH)

```

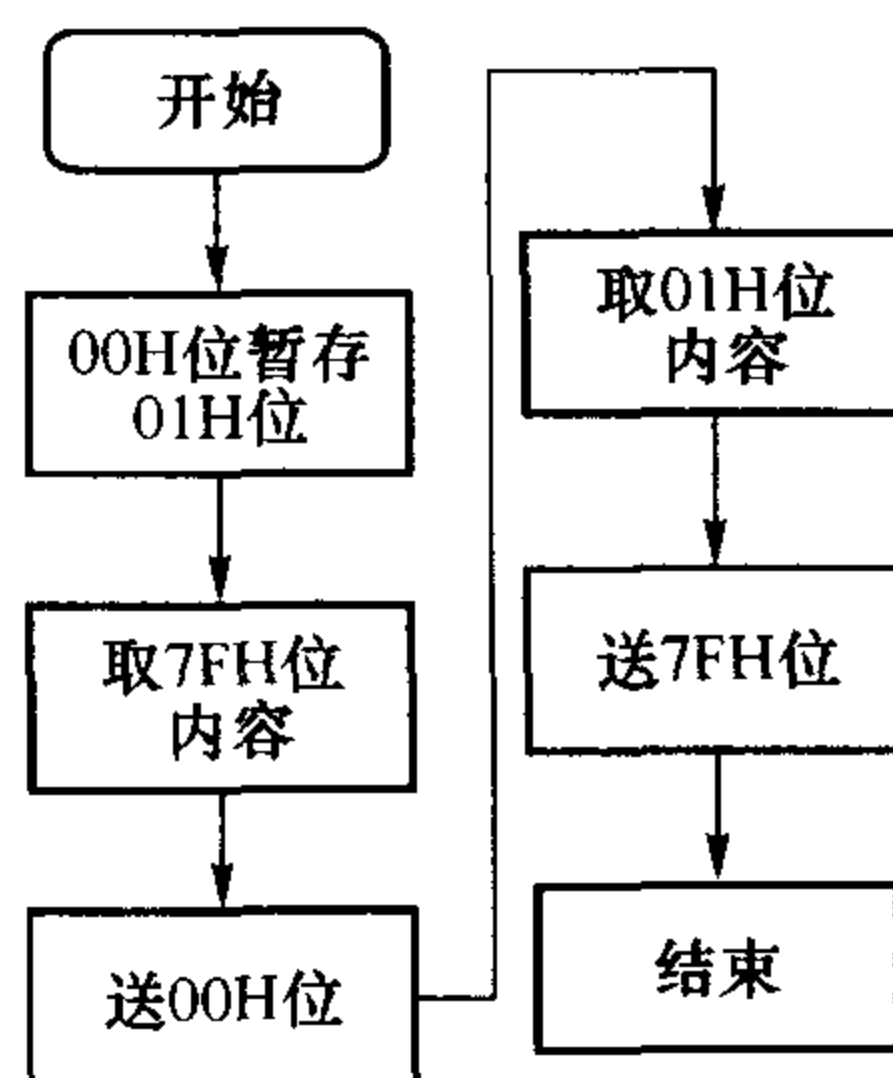


图 4.3.5 例 4-38 程序流程图

## 2. 位清 0, 置 1, 取反指令, CLR, SETB(Set Bit), CPL

```

CLR    C          ; 0→(Cy)
CLR    bit        ; 0→(bit)
SETB   C          ; 1→(Cy)
SETB   bit        ; 1→(bit)
CPL    C          ; (Cy)→(Cy)
CPL    bit        ; (bit)→(bit)

```

这组指令的功能分别是清 0、置 1 和取反进位标志或直接寻址位。不影响其他寄存器或标志位。当直接位地址为端口中某一位时，具有“读—修改—写”功能。

**例 4-39** (21H)=71H, Cy=1, 顺序执行以下指令, 分析结果及 Cy。

程序	执行结果
CLR C	; (Cy) = 0
CLR 08H	; 08H 为 21H 的 D0 位, (21H) = 70H
CPL 09H	; 09H 为 21H 的 D1 位, (21H) = 72H
SETB C	; (Cy) = 1
SETB 0FH	; 0FH 为 21H 的 D7 位, (21H) = 0F2H
CPL C	; (Cy) = 0

所以, 程序执行完成后, (21H)=0F2H, Cy=0。21H 中的位定义如图 4.3.6 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H

图 4.3.6 21H 中的位定义

## 3. 位运算指令, ANL, ORL

```

ANL    C,    bit    ; (Cy) ∧ (bit)→(Cy)
ANL    C,    /bit   ; (Cy) ∧ (bit)→(Cy)
ORL    C,    bit    ; (Cy) ∨ (bit)→(Cy)
ORL    C,    /bit   ; (Cy) ∨ (bit)→(Cy)

```

这组指令的功能分别是进位标志 Cy 的内容与直接位地址的内容进行逻辑与、或操作,结果送 Cy。其中斜杠“/”表示对该位取反后再参与运算,但不改变原来数值。

4. 位条件转移指令,JC (Jump if Carry Flag Set),JNC (Jump if No Carry Flag),JB (Jump if Direct Bit Set),JNB (Jump if Direct Not Set),JBC (Jump if Direct Set & Clear Bit)

JC	rel	;若(Cy) = 1,则(PC) + 2 + rel → (PC) ;若(Cy) = 0,则(PC) + 2 → (PC)
JNC	rel	;若(Cy) = 0,则(PC) + 2 + rel → (PC) ;若(Cy) = 1,则(PC) + 2 → (PC)
JB	bit, rel	;若(bit) = 1,则(PC) + 3 + rel → (PC) ;若(bit) = 0,则(PC) + 3 → (PC)
JNB	bit, rel	;若(bit) = 0,则(PC) + 3 + rel → (PC) ;若(bit) = 1,则(PC) + 3 → (PC)
JBC	bit, rel	;若(bit) = 1,则(PC) + 3 + rel → (PC),且(bit) = 0 ;若(bit) = 0,则(PC) + 3 → (PC)

这组指令的功能是若满足条件则转移到目的地址去执行,不满足条件则顺序执行下一条指令。指令执行过程如图 4.3.7 所示。注意:目的地址一定要在以下一条指令起始地址为中心的 256 B 范围内(−128~+127 B)。

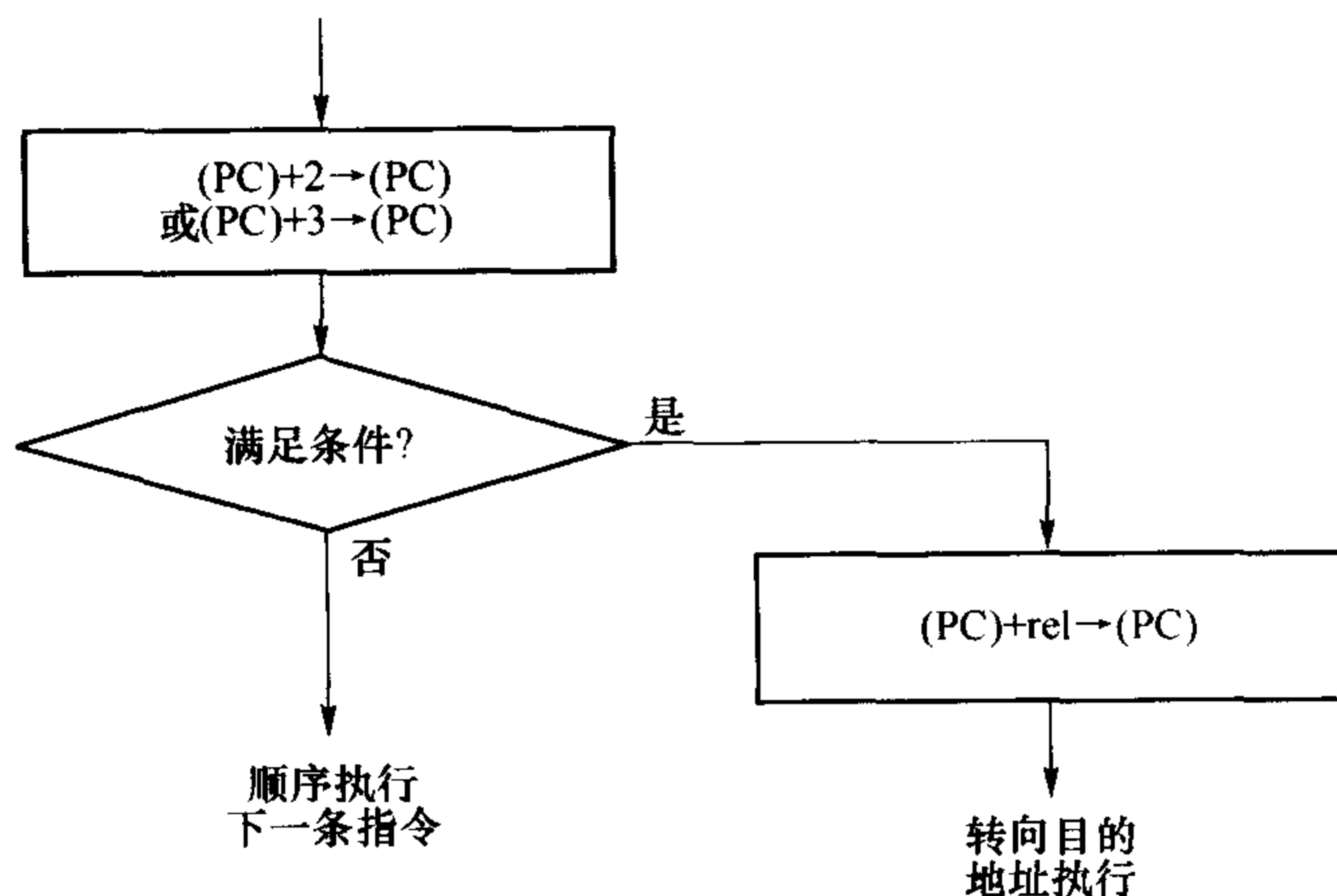


图 4.3.7 位变量条件转移指令执行流程图

**例 4-40** 20H 的 D0 位存放测量数据超限标志,编写程序,使 D0=1 时 P1.0 输出高电平报警,当 D0=0 时 P1.0 输出低电平取消报警。

解

```

ALARM: JB    00H, ALARM1
        CLR   P1.0          ;P1.0 输出低电平
        RET
ALARM1: SETB  P1.0          ;P1.0 输出高电平
        RET
  
```



### 4.3.5 控制转移类指令

程序在执行过程中,有时因为操作的需要,不能按顺序逐条执行指令,而需要改变程序的运行方向,即将程序跳转到某个指定的地址处再执行,也就是需要指令具有修改程序计数器 PC 内容的功能。完成这些操作需要利用控制转移类指令。助记符为: AJMP、LJMP、SJMP、JMP、JZ、JNZ、CJNE、DJNZ、ACALL、LCALL、RET、RETI、NOP。

1. 无条件转移指令, AJMP (Absolute Jump), LJMP (Long Jump), SJMP (Short Jump), JMP (Jump)

```
AJMP    addr11    ; 双字节, 双周期, (PC) + 2 → (PC), addr11 → (PC10~0)
LJMP    addr16    ; 三字节, 双周期, addr16 → (PC)
SJMP    rel       ; 双字节, 双周期, (PC) + 2 → (PC), (PC) + rel → (PC)
JMP     @A + DPTR ; 单字节, 双周期, (A) + (DPTR) → (PC)
```

此组指令无条件转移到指定的目的地址,区别是目的地址的范围和计算方法不同。

第一条为绝对短转移指令,11 位地址可变( $2^{11}=2048=2\text{ KB}$ ),所以转移范围为 2 KB ( $\times 000\text{H} \sim \times 7\text{FFH}$  或  $\times 800\text{H} \sim \times \text{FFFH}$ )空间,2 KB 一页,64 KB 程序存储器共 32 页。转移的目的地址必须与 AJMP 指令的下一条指令的第一个字节在同一 2 KB 的范围内,否则转移出错。在用 KEIL C 进行模拟调试时,自动计算地址是否超出范围,若超出范围则提示 TARGET OUT OF RANGE。

**例 4-41** 求下列程序中目的地址 JMPSUB 的范围。

```
ORG     1FF0H
TEST:   AJMP    JMPSUB
```

程序的执行过程如下。

(1) 先计算 PC:  $\text{PC} = 1\text{FF}0\text{H} + 02\text{H} = 1\text{FF}2\text{H}$ ;

(2) 求 PC 的高 5 位:  $\text{PC}_{15\sim 11} = 00011$ ;

(3) 目的地址范围:  $0001100000000000\text{B} \sim 0001111111111111\text{B}$ , 即  $1800\text{H} \sim 1\text{FFFH}$ 。

转移的目的地址必须在此范围,否则出错。

第 2 条为长转移指令,16 位地址可变( $2^{16}=65536=64\text{ KB}$ ),所以转移范围为 64 KB 空间。转移的目的地址可以在 64 KB 程序存储器地址空间的任何地方,不影响任何标志位。不论长转移指令放在程序的什么位置,程序执行时都会转移到相应的 16 位地址处执行。

第 3 条为相对短转移指令。地址偏移量 rel 是 8 位带符号数,用补码表示,其值范围为  $-128 \sim +127$ 。当 rel 为正数时,表示正向转移;为负数时,表示反向转移。

第 4 条为间接转移指令,累加器 A 中的 8 位无符号数与数据指针 DPTR 中的 16 位地址相加,相加结果的 16 位新地址(即转移目的地址)送 PC。不改变累加器和数据指针内容,也不影响标志位。常用于散转程序。

**例 4-42** 已知键值 0~3,存于 30H 中,不同的键值,执行不同的功能,各功能起始地址为 KEY0~KEY3,编写散转程序。

解

```

MOV    A,    30H    ;取键值
RL     A           ;键值乘 2
MOV    DPTR, #JMPTAB ;转移指令表首址
JMP    @A + DPTR    ;转入转移指令表
      ⋮
JMPTAB: AJMP   KEY0    ;键值 0 功能程序,转移地址:JMPTAB
        AJMP   KEY1    ;键值 1 功能程序,转移地址:JMPTAB + 2
        AJMP   KEY2    ;键值 2 功能程序,转移地址:JMPTAB + 4
        AJMP   KEY3    ;键值 3 功能程序,转移地址:JMPTAB + 6

```

注意 RL A 指令完成键值乘 2 功能,原因是 AJMP 指令占两个字节,保证正确转移。如 30H 中为 2 时,移位后  $A=4$ ,转移地址为  $JMPTAB+4$ ,散转后执行 AJMP KEY2 指令。

## 2. 条件转移指令

该组指令包括累加器 A 判零转移指令、比较转移指令、减一条件转移指令。

(1) 累加器 A 判零转移指令, JZ (Jump if Accumulator is Zero), JNZ (Jump if Accumulator is Not Clear)

JZ rel ;若  $(A)=0$ ,  $(PC)+2+rel \rightarrow (PC)$ , 若  $(A) \neq 0$ , 则  $(PC)+2 \rightarrow (PC)$

JNZ rel ;若  $(A) \neq 0$ ,  $(PC)+2+rel \rightarrow (PC)$ , 若  $(A)=0$ , 则  $(PC)+2 \rightarrow (PC)$

这组指令为双字节指令,条件满足,则转移到 rel 指定的目的地址,条件不满足,则继续执行下一条指令,如图 4.3.8 所示。目的地址在以下一条指令的起始地址为中心的 256 B 范围内( $-128 \sim +127$ )。指令的执行不改变累加器 A 中的内容,不影响标志位。

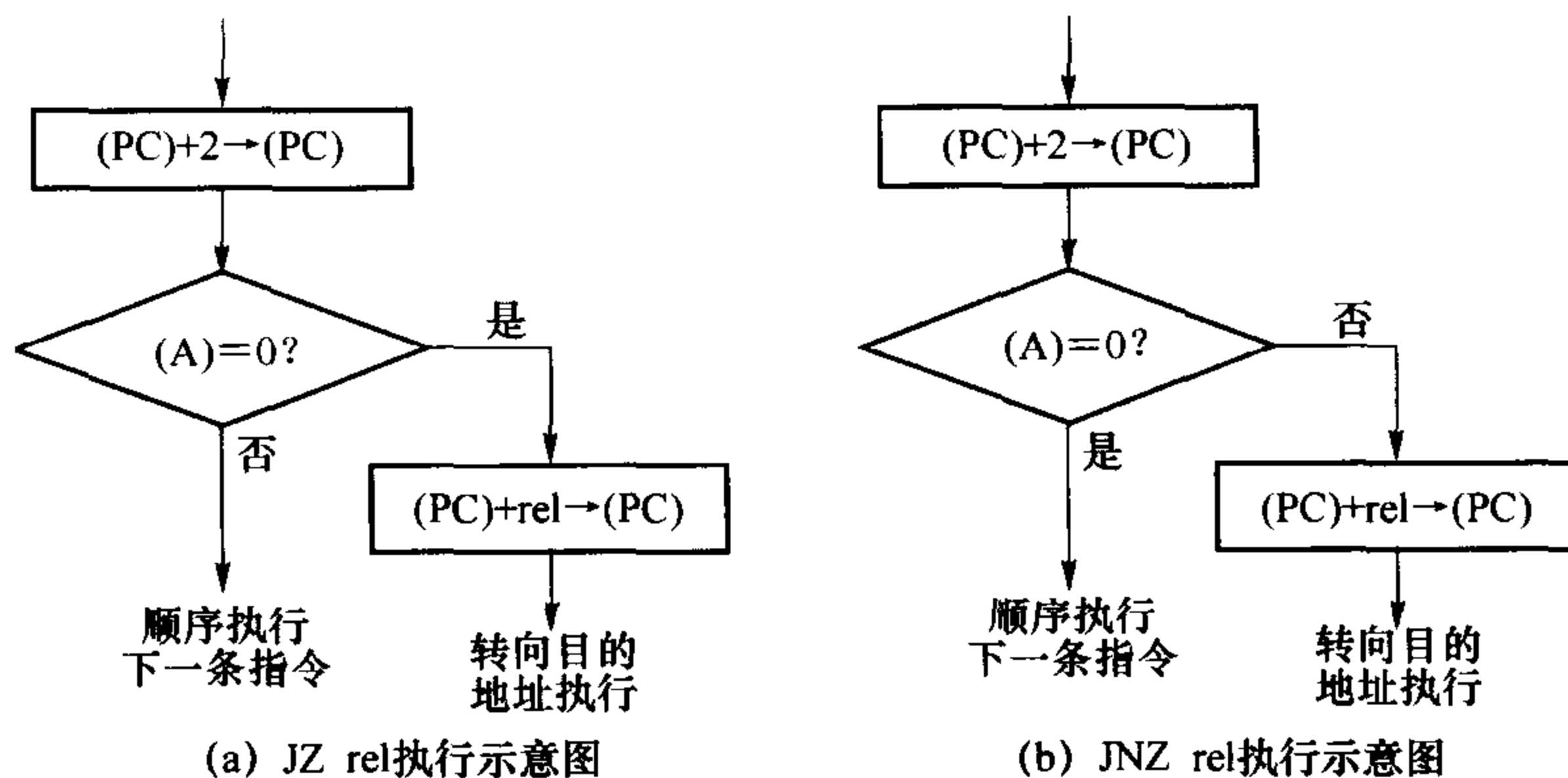


图 4.3.8 JZ 和 JNZ 指令的逻辑流程图

**例 4-43** 设  $(A)=08H$ , 执行下列指令, 分析程序转移情况。

```

      ⋮
JZ    LOOP1 ;  $(A) \neq 0$ , 程序继续执行
CLR   A     ;  $(A) = 0$ 

```

JZ     LOOP2   ; (A) = 0, 程序转向 LOOP2 执行  
 ⋮

(2) 比较转移指令, CJNE (Compare Jump if Not Equal)

CJNE A,    direct, rel  
 CJNE A,    #data, rel  
 CJNE Rn,   #data, rel  
 CJNE @Ri, #data, rel

这组指令为三字节指令, 功能是比较两个操作数的大小, 数值不相等时转移, 相等则继续执行下一条指令。转移时根据操作数内容的大小, 改变进位标志 Cy。具体过程如下:

- ① 若目的操作数=源操作数, 则  $(PC)+3 \rightarrow (PC)$ ,  $Cy=0$ ;
- ② 若目的操作数>源操作数, 则  $(PC)+3+rel \rightarrow (PC)$ ,  $Cy=0$ ;
- ③ 若目的操作数<源操作数, 则  $(PC)+3+rel \rightarrow (PC)$ ,  $Cy=1$ 。

CJNE 指令流程示意图如 4.3.9 所示。

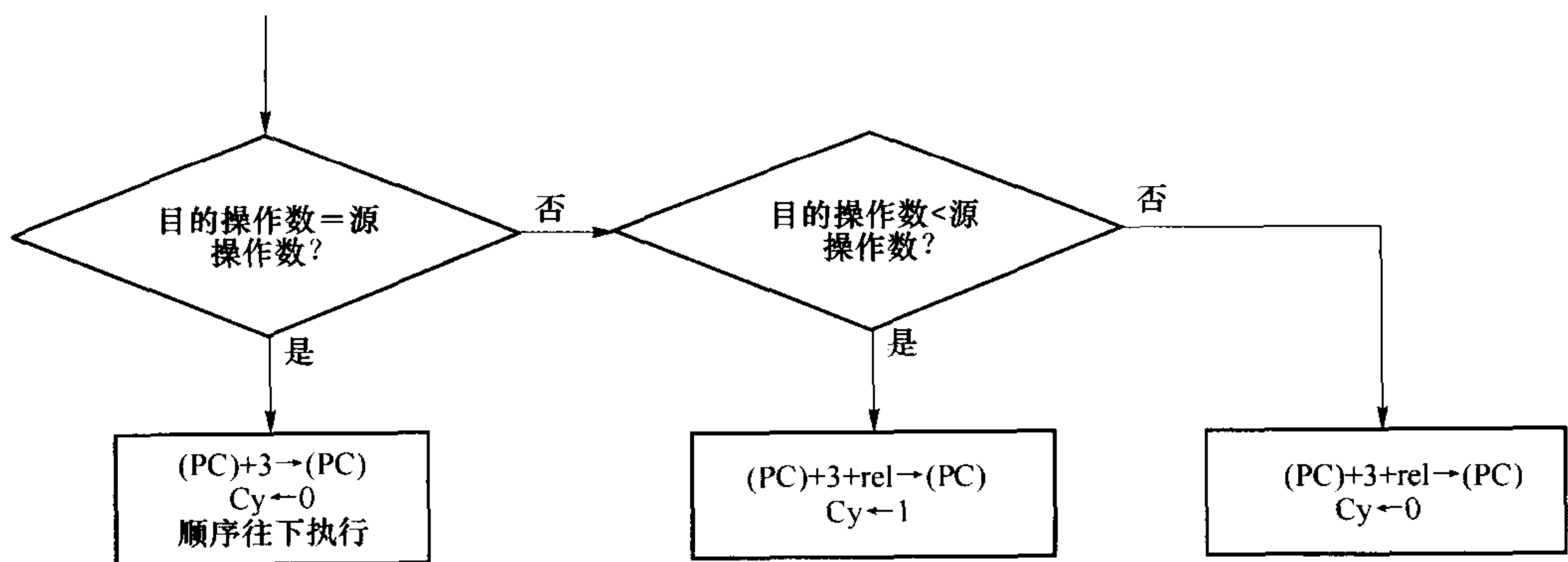


图 4.3.9 CJNE 指令流程示意图

**例 4-44** 编写程序根据 A 中内容大于 40H、等于 40H、小于 40H 分别执行不同的功能。

**解**

```

CJNE A,    #40H, NEQ   ; (A) ≠ 40H, 程序转移
EQ:     ...             ; (A) = 40H, 处理程序
       ⋮
NEQ:    JC   SMALL       ; (A) < 40H, 程序转移
       ⋮               ; (A) > 40H, 处理程序
SMALL:  ...             ; (A) < 40H, 处理程序
  
```

**例 4-45** 求 R2、R3 中的较大值, 并将较大值保存在 R7 中, 较小值保存在 R6 中。

**解** 编写程序时, 经常将程序写为子程序的形式, 方便程序的调用, 提高程序的利用率。编写子程序时, 应规定好入口参数和出口参数以及子程序的名称。

入口参数: R2, R3; 出口参数: R7, R6; 名称: COMPARE。

```

        ORG      0500H
COMPARE: MOV      A, R2          ; (R2) → (A)
        MOV      30H, R3        ; (R3) → (30H), 30H 为暂存器
        CJNE     A, 30H, NEQ
NEQ:    JC        SAVE
        XCH      A, 30H
SAVE:   MOV      R6, A
        MOV      R7, 30H
        RET

```

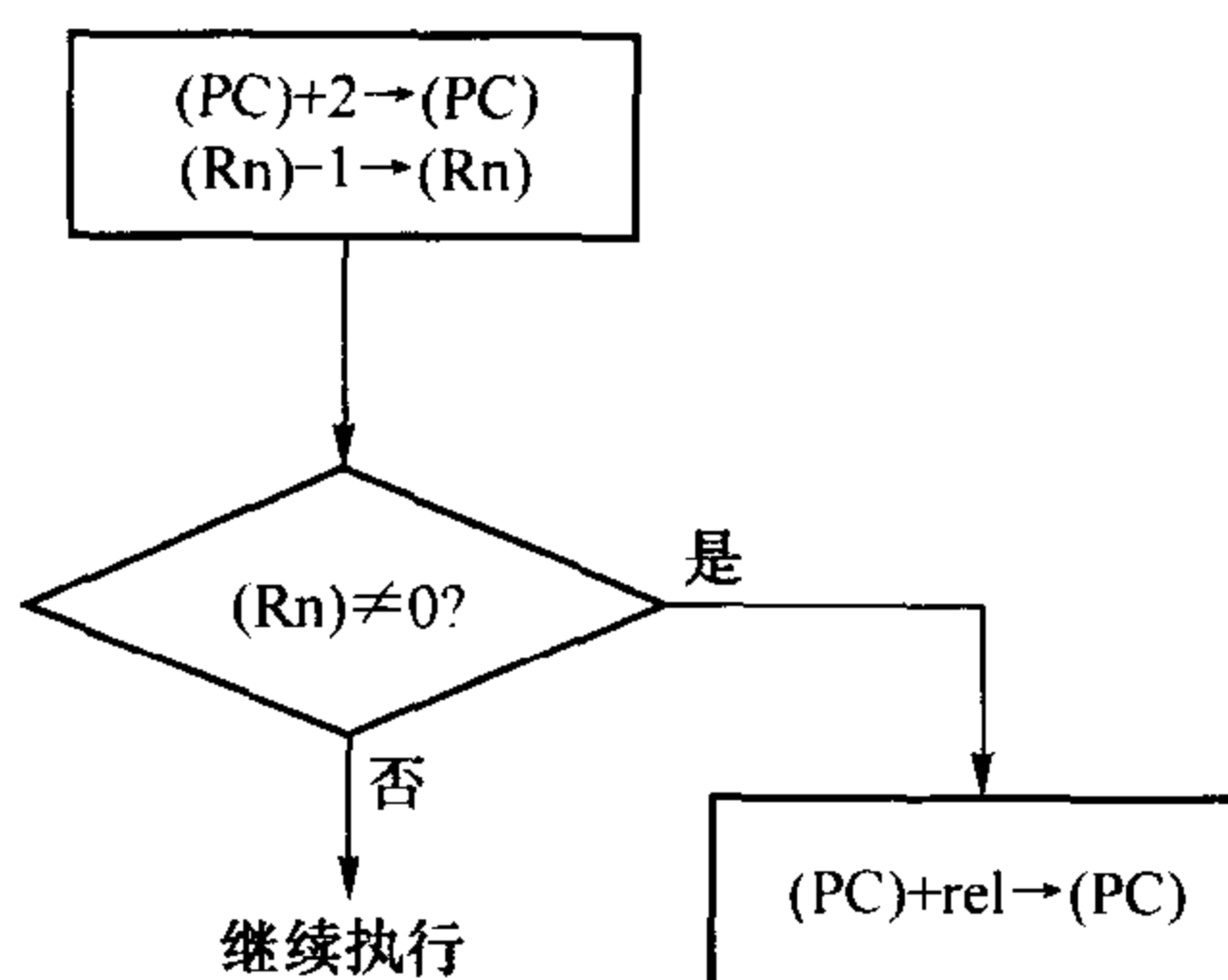
(3) 减一条件转移指令, DJNZ (Decrement Jump if Not Zero)

```

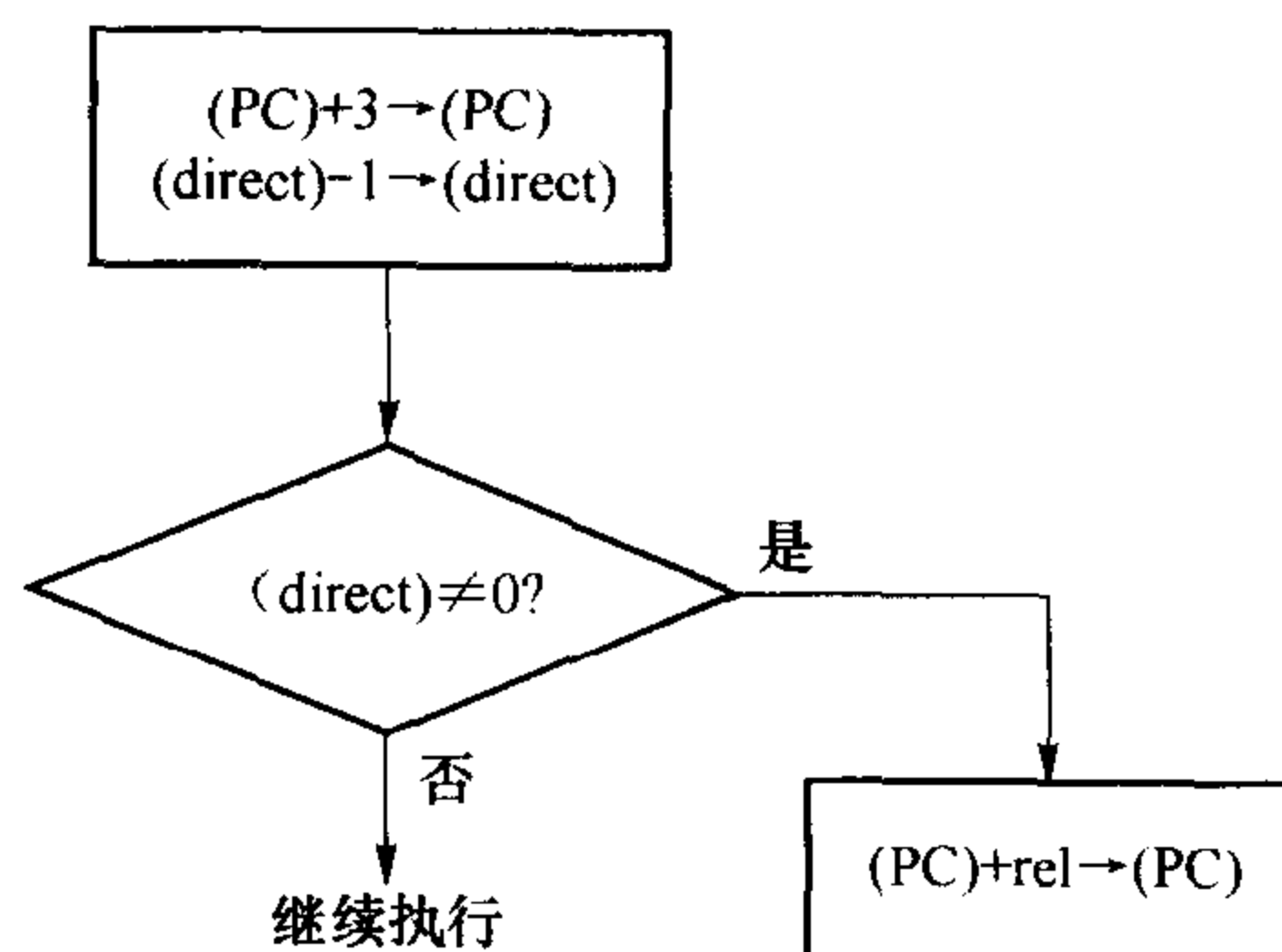
DJNZ    Rn, rel
DJNZ    direct, rel

```

这组指令的功能是指令每执行一次, 将目的操作数所指向的地址单元的内容减 1, 然后判断其值是否为 0。若不为 0, 则程序转移到目的地址继续执行; 若为 0, 则按顺序继续往下执行。执行流程如图 4.3.10 所示。



(a) DJNZ Rn, rel 执行示意图



(b) DJNZ direct, rel 执行示意图

图 4.3.10 DJNZ 指令执行流程图

**例 4-46** 从 P1.7 输出 5 个方波。

**解**

```

MOV     R2, #0AH    ; 方波个数初值
CLR     P1.7        ; 清 P1.7
PULSE:  CPL     P1.7 ; P1.7 位状态取反
DJNZ    R2, PULSE   ; (R2)-1 ≠ 0, 继续输出, (R2)-1 = 0, 退出循环结束输出
        ⋮

```

方波的高电平或低电平时间为 4 个机器周期(CPL 指令为 1 个机器周期;DJNZ 指令为 3 个机器周期),具体时间由 CPU 的晶振决定。图 4.3.11 为方波输出图形。

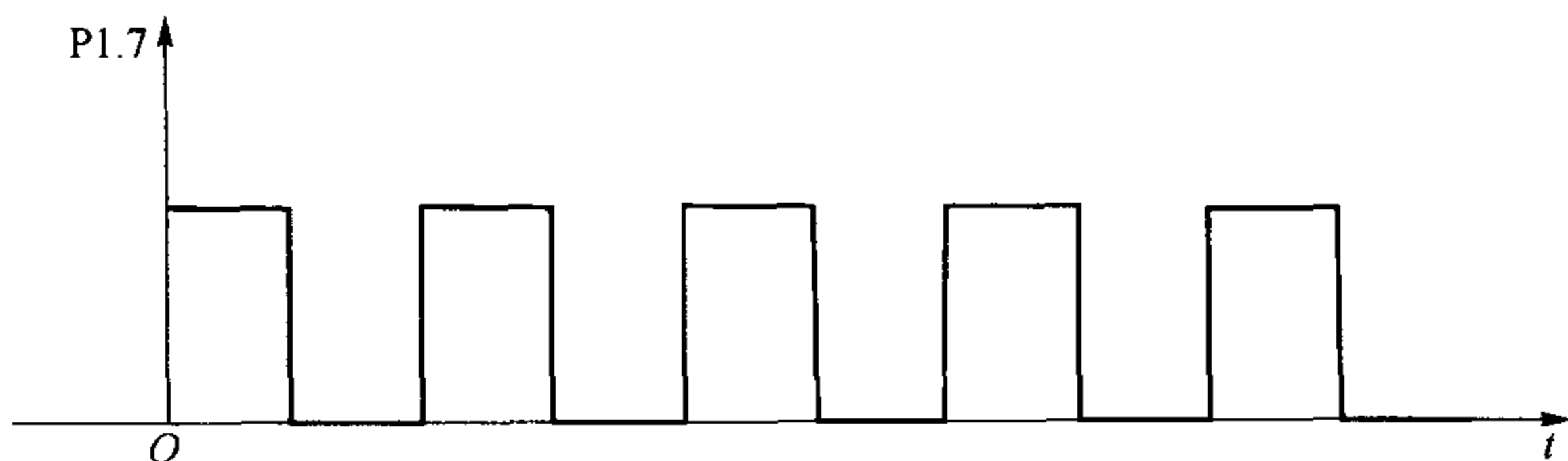


图 4.3.11 P1.7 输出波形示意图

### 3. 子程序调用和返回指令

编写程序时,往往许多地方需要执行同样的操作或运算,如压缩 BCD 码变成分离 BCD 码、双字节加法程序等,这时可以把这些多次使用的程序段从整个程序中独立出来,单独编写成一个公用程序段,这种相对独立、具有一定功能的公用程序称为子程序。调用子程序的程序称为主程序。子程序的最后一条指令为返回主程序的指令(RET)。主程序通过调用指令(ACALL,LCALL)自动转入子程序,主程序调用子程序以及从子程序返回主程序的过程如图 4.3.12 所示。

当主程序执行到 A 处,遇到调用子程序 ADDSUB 指令时,CPU 首先自动把 B 处(称为断点)即调用指令下一条指令第一字节的地址(PC 值)保留到堆栈中(自动压栈),然后将子程序 ADDSUB 的起始地址送入 PC,于是,CPU 转向执行子程序 ADDSUB;当程序执行到 RET 指令时,CPU 自动把断点 B 处的地址从堆栈中弹出送 PC,保证 CPU 回到主程序继续执行。当程序执行到 C 处再次遇到调用子程序 ADDSUB 指令时,重复上述过程。

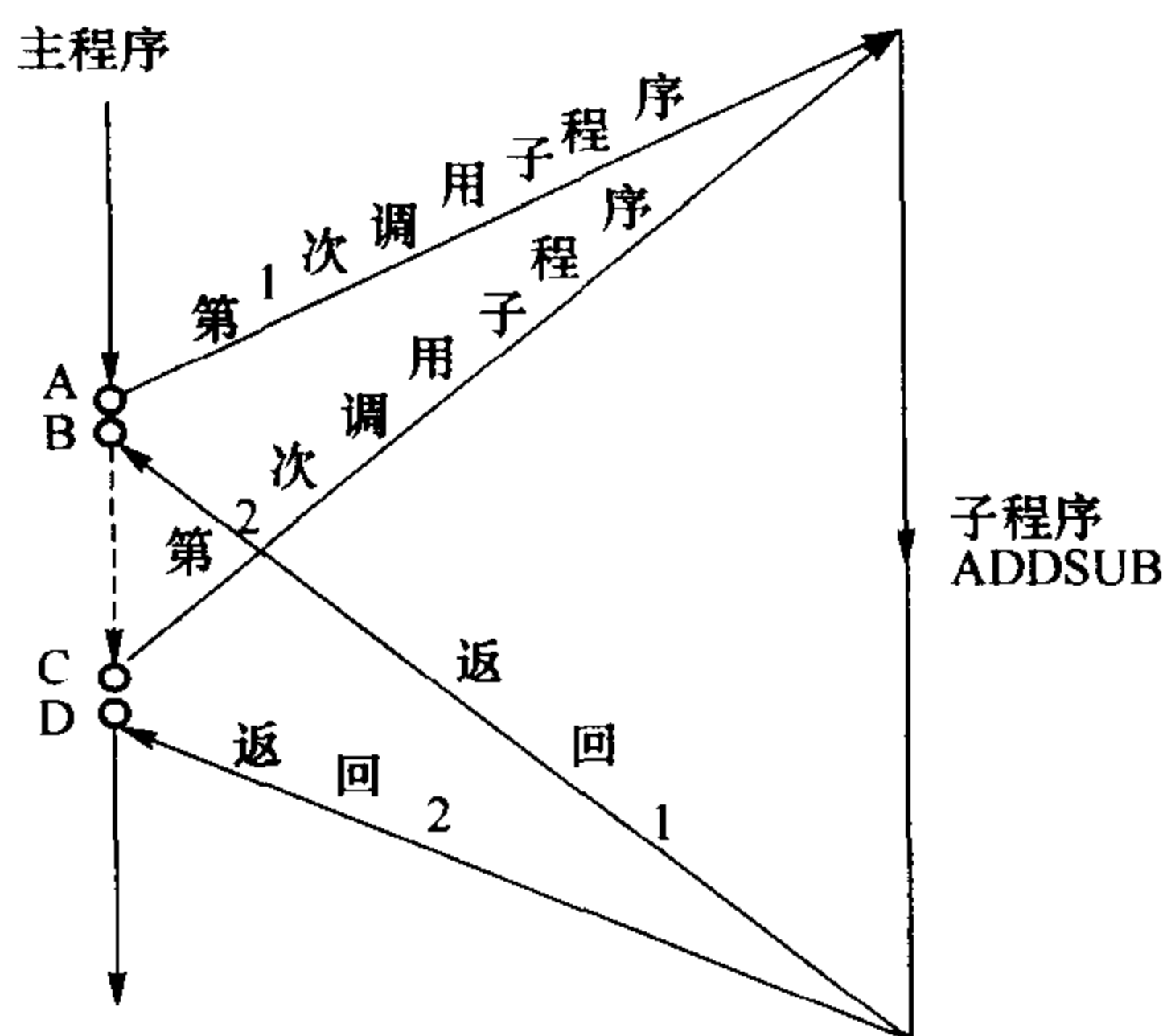


图 4.3.12 主程序调用子程序及返回示意图

归纳子程序调用过程如下:自动压栈保存断点 PC→子程序地址送 PC→执行子程序→子程序遇 RET 指令→自动弹栈恢复断点 PC→返回主程序→继续执行主程序。

子程序调用过程就如读书读完 50 页时,有其他事情要做,放一支书签在当前页,事情做完,打开书找到书签位置继续读 51 页。书签起到了保存读书状态和恢复读书状态的作用,保证按顺序读书、不会漏读。如果没有书签,则是一团糟的状态。

主程序和子程序是相对的,一个子程序调用另一个子程序时,就变成了另一个子程序的主程序,称为子程序的嵌套。图 4.3.13 表示一个两级子程序嵌套的子程序调用及堆栈中断点地址的存放情况。

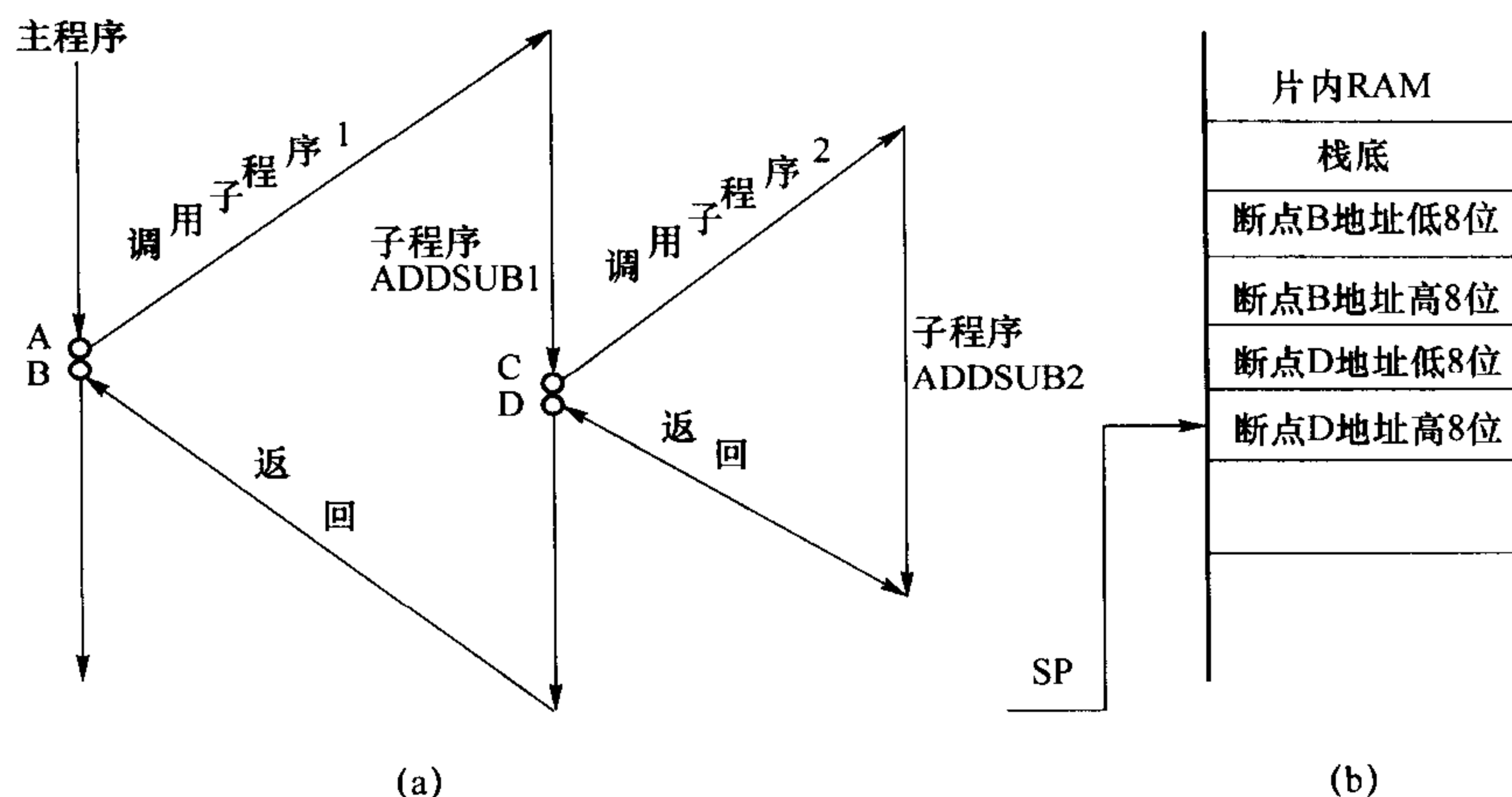


图 4.3.13 两级子程序嵌套及断点地址的存放

子程序的优点:减少编写和调试工作量,减少程序占用的存储空间,提高程序利用率,简化程序结构。编写子程序时,应写清入口参数和出口参数,方便调用,详见 5.3.4 小节。

(1) 调用指令,ACALL (Absolute Subroutine Call),LCALL (Long Subroutine Call)

ACALL addr11 ;双字节,双周期, $(PC) + 2 \rightarrow (PC)$   
 $; (SP) + 1 \rightarrow (SP), PC_{7 \sim 0} \rightarrow (SP)$   
 $; (SP) + 1 \rightarrow (SP), PC_{15 \sim 8} \rightarrow (SP)$   
 $; addr11 \rightarrow PC_{10 \sim 0}, PC$  高 5 位不变

LCALL addr16 ;三字节,双周期, $(PC) + 3 \rightarrow (PC)$   
 $; (SP) + 1 \rightarrow (SP), PC_{7 \sim 0} \rightarrow (SP)$   
 $; (SP) + 1 \rightarrow (SP), PC_{15 \sim 8} \rightarrow (SP)$   
 $; addr16 \rightarrow (PC)$

此组指令为子程序调用指令,隐含两次压栈操作,压栈时先压 PC 低 8 位,后压 PC 高 8 位。执行过程如下:首先自动计算调用指令下一条指令的起始地址,第一条指令为双字节指令,PC 值加 2,第二条指令为三字节指令,PC 值加 3;其次将计算出的 PC 值的低 8 位和高 8 位分别压栈;最后将子程序的入口地址送 PC,CPU 转向子程序开始执行。

ACALL 指令为 2 KB 范围内的短调用指令,用法与 AJMP 类似。所调用的目的地址必须与 ACALL 的下一条指令的第一个字节在同一 2 KB 范围内,这是因为调用的目的地址与 ACALL 的下一条指令的第一个字节的高 5 位  $addr_{15 \sim 11}$  相同。LCALL 指令为长调用指令,子程序的首地址可以设置在 64 KB 程序存储器地址空间的任何位置。

**例 4-47**  $(SP) = 60H$ ,分析下列程序的执行过程,并求子程序 SUBADD 的目的地址范围。

```
ORG      2FFEH
TEST: ACALL SUBADD
```

**解** 程序的执行过程如下。

① 先计算 PC:  $PC = 2FFEH + 02H = 3000H$ ;

② 低 8 位入栈:  $(SP) = 60H + 1H = 61H, (61H) = 00H$ ;

③ 高 8 位入栈:  $(SP) = 61H + 1H = 62H, (62H) = 30H$ ;

④ 求 PC 的高 5 位:  $PC_{15\sim11} = 00110$ ;

⑤ 目的地址范围:  $0011000000000000 \sim 0011011111111111$ , 即  $3000H \sim 37FFH$ 。调用的目的地址必须在此范围, 否则将出错。

**例 4-48** 将例 4-47 中程序改为长调用指令, 分析 SP 中的内容。

**解** 程序的执行过程如下。

① 先计算 PC:  $PC = 2FFE H + 03H = 3001H$ ;

② 低 8 位入栈:  $(SP) = 60H + 1H = 61H, (61H) = 01H$ ;

③ 高 8 位入栈:  $(SP) = 61H + 1H = 62H, (62H) = 30H$ ;

④ 目的地址范围:  $0000H \sim FFFFH$ 。

(2) 返回指令, RET (Return from Subroutine), RETI (Return from Interrupt)

RET ;  $((SP)) \rightarrow PC_{15\sim8}$ , 弹出 PC 高 8 位,  $(SP) - 1 \rightarrow (SP)$

;  $((SP)) \rightarrow PC_{7\sim0}$ , 弹出 PC 低 8 位,  $(SP) - 1 \rightarrow (SP)$

RETI ;  $((SP)) \rightarrow PC_{15\sim8}$ , 弹出 PC 高 8 位,  $(SP) - 1 \rightarrow (SP)$

;  $((SP)) \rightarrow PC_{7\sim0}$ , 弹出 PC 低 8 位,  $(SP) - 1 \rightarrow (SP)$

返回指令把堆栈中的断点地址恢复到程序计数器 PC 内, 使程序回到断点处继续执行。这组指令隐含两次弹栈操作, 弹栈时先弹 PC 高 8 位, 后弹 PC 低 8 位。

RET 为子程序返回指令, 只能用在子程序末尾。

RETI 为中断返回指令, 只能用在中断程序末尾。执行该指令后, 除程序返回原断点地址处继续执行外, 还清除相应的中断优先级状态位, 以允许 CPU 响应低优先级的中断请求。CPU 执行 RETI 指令后至少需要再执行一条指令, 才能响应新的中断请求。

**例 4-49** 编写程序将内部 RAM 的  $30H \sim 3AH$ 、 $40H \sim 4EH$ 、 $50H \sim 5FH$  区域清 0。

**解**

```

                ORG      0000H                ;CPU 复位后 PC 值
                LJMP MAIN                    ;转向 MAIN 程序
                ORG 0100H                    ;MAIN 起始地址
MAIN:  MOV      SP,      #60H                ;设置堆栈栈底地址指针为 60H
        MOV      R0,      #30H                ;第一清 0 区首址送 R0
        MOV      R2,      #0BH                ;第一清 0 区单元个数送 R2
        LCALL    ZERO                        ;30H~3AH 单元清 0
        MOV      R0,      #40H                ;第二清 0 区首址送 R0
        MOV      R2,      #0FH                ;第二清 0 区单元个数送 R2
        LCALL    ZERO                        ;40H~4EH 单元清 0
        MOV      R0,      #50H                ;第三清 0 区首址送 R0
        MOV      R2,      #10H                ;第三清 0 区单元个数送 R2
        LCALL    ZERO                        ;50H~5FH 单元清 0
        ...

```



```

ZERO:  MOV    @R0,    #00H    ;清 0
        INC     R0           ;修改清 0 区地址指针
        DJNZ    R2,     ZERO    ; (R2) - 1 ≠ 0, 转向 ZERO
        RET              ; (R2) - 1 = 0, 清 0 完毕, 子程序返回

```

#### 4. 空操作指令

```

NOP                      ; (PC) + 1 → (PC), 单字节

```

执行该单字节单周期指令仅使程序计数器 PC 值加 1, 不进行任何操作, 消耗时间为 12 个时钟周期, 可作短时间的延时。如例 4-46, 当略增方波周期时, 可采用 NOP 指令。其程序清单为:

```

        MOV     R2, #0AH      ;方波个数初值
        CLR     P1.7          ;清 P1.7
PULSE:  CPL     P1.7          ;P1.7 位状态取反
        NOP                      ;延时
        NOP                      ;延时
        DJNZ    R2, PULSE      ;(R2) - 1 ≠ 0, 继续输出; (R2) - 1 = 0, 退出循
                                ;环结束输出
        :

```

方波的高电平或低电平时间为 6 个机器周期 (CPL 指令 1 个机器周期; DJNZ 指令 3 个机器周期, NOP 指令 1 个指令周期)。相对例 4-46 的方波周期加大。

本章主要介绍 AT89S52 汇编语言指令系统, 指令是编程设计的基础, 所以必须很好地掌握指令格式、执行过程、周期、字节数。指令的机器码参见附录 1 和附录 2。

## 习 题

1. 写出汇编执行指令的格式, 并解释各部分的含义。
2. 简述汇编伪指令的特点, 说明 ORG, DB, END, EQU, DS 的用法和功能并举例。
3. 什么是寻址? 为什么要寻址? AT89S52 共有几种寻址方式?
4. 变址间接寻址有何特点? 主要应用在什么场合? 采用 DPTR 或 PC 作为基址寄存器其寻址范围有何不同?
5. 对 AT89S52 片内高 128 B 地址空间寻址时应注意什么? 如何实现?
6. 访问程序存储器使用哪些寻址方式? 哪些指令可以实现?
7. 请指出下列指令的区别, 并说明寻址方式。

```

MOV     A,        60H
MOV     A,        #60H
MOV     60H,      #40H
MOV     60H,      40H

```

8. 分析下列程序执行后, RAM 的 0B0H 单元及 SFR 的 P3 口内容。

```

MOV     A,        #58H

```

```
MOV    R0,        #0B0H
```

```
MOV    @R0,       A
```

```
MOV    0B0H,      #28H
```

9. 访问内部 RAM 单元可使用哪些寻址方式?
10. 访问外部 RAM 单元可使用哪些寻址方式?
11. 编写程序将内部 RAM 单元的 50H 的内容送外部 RAM 的 7FFEh 单元。
12. 编写程序将内部 RAM 单元的 0A0H 的内容送外部 RAM 的 07FEh 单元。
13. 编写查共阳极数码管的显示代码程序,要显示的数据存在 30H 单元。
14. 已知 R6,R7 中有一个 16 位二进制数,高位在 R6,低位在 R7,请编程将其求补,结果存回原处。
15. 已知 40H,41H 中有一个 16 位二进制数,高位在 40H,低位在 41H,请编程将其乘 2,结果存回原处。
16. 已知减数存放在 R3,R4 中,被减数存放在 R5,R6 中,高位字节在 R3,R5,低位字节在 R4,R6,编写双字节减法程序,结果存于 32H,33H。
17. 编程求 4 字节 BCD 码的和,加数在 30H~33H 单元,被加数在 40H~43H 单元,和保存在 33H~37H 单元。
18. 利用位操作实现下列逻辑操作,不能改变未涉及位的内容。
  - (1) P2 口的 P2.0,P2.1 置位;
  - (2) ACC.7,ACC.5,ACC.3,ACC.1 清除;
  - (3) 清除累加器的低 4 位。
19. 编程将工作寄存器组三区的内容传递到 20H 开始的单元。
20. 编程将内部 RAM 的 50H~55H 的内容依次存入 5FH~5AH。
21. 编程查找在内部 RAM 单元的 20H~50H 单元是否有 0DH 这一数据,若有将 00H 位置 1,否则清 0。
22. 编程查找在内部 RAM 单元的 20H~50H 单元出现数据 0DH 的次数,并将查找结果存入 51H 单元。
23. 编写程序将片外 RAM 的 3000H 单元开始存放的 20 个数传送到片内 30H 开始的单元。
24. 已知 30H 单元存放的字符可能是 \$、#、%、!,内容不同转入相应的处理程序,请设计其程序。
25. 求下列程序转移目的地址 DISPLAY、ADDN 的范围。

```
ORG    2FF0H
```

```
TEST:  AJMP    DISPLAY
```

```
TEST1: LJMP    ADDN
```

26. 分析下列程序执行时 SP、PC 中的数值,以及堆栈区的内容。

```
ORG      2000H
MOV      SP,      # 50H
LCALL    SUBADD
MOV      A,      # 30H
        ⋮
ORG      3000H
SUBADD:  MOV      R0,      # 30H
        LCALL    BIN2BCD
        RET
BIN2BCD: ⋮
```

## 第5章 AT89S52 程序设计与调试

学习单片机指令系统的目的是能够将其汇编指令有机地组合在一起,根据硬件结构设计出满足实际需要的完整、可靠、可灵活扩展的应用软件。汇编语言是一种面向机器的语言,与面向过程的高级语言相比,其缺点是编程不够方便,不易移植到其他类型机器;但其显著的优点是程序结构紧凑,占用存储空间小,实时性强,执行速度快,能直接管理和控制存储器及硬件接口,充分发挥硬件的作用,因而特别适用于编写实时测控、软硬件关系密切、程序编制工作量不太大的单片机应用系统的程序开发。对于从事单片机应用系统的程序设计开发的人员来说,首先必须掌握编程的步骤和方法、汇编程序的基本格式,其次应该熟练应用单片机系统的开发、调试工具,并最终将程序下载到单片机中独立运行。本章主要介绍以上几个方面的内容,目的是通过本章学习,编程人员能够顺利编写出完成一定功能的程序,并能很好地利用相应的开发系统进行调试、下载到单片机中独立运行。另外,本章用一节内容介绍 C 语言应用于单片机编程的有关实用知识,目的是希望读者能够了解 C 语言编写单片机应用程序的方法、步骤,并能够熟练应用汇编语言和 C 语言进行系统开发。

### 5.1 程序设计步骤

根据设计任务要求,采用汇编语言编写程序的过程称为汇编语言程序设计。对于一个单片机应用系统,在硬件调试通过后便可着手进行应用程序的开发。开发一个完整的应用程序大致可分为以下几个步骤。

#### 1. 拟定设计任务书

即根据实际系统的功能要求,写出软件具体实现的功能,如测量数据显示,日期、时间显示,故障信息显示,与计算机进行通信等。设计任务书要条理清楚、内容完善。

#### 2. 建立数学模型并确定算法

根据实际情况,描述出各输出变量与输入变量之间的数学关系,即数学模型。数学模型的正确度是系统性能好坏的决定性因素之一。例如在热电偶温度测量控制系统中,要得到当前的温度值,需要根据热电偶的温度-电压之间的数学模型来计算。而热电偶的输出电压与温度的关系为非线性关系,这就涉及如何将其进行线性化处理的问题,线性化处理对系统的测量精度起决定性的作用,也直接关系到系统的控制精度。所以系统的数学模型和根据数学模型确定的算法一定要统筹考虑,认真确定。

#### 3. 程序的总体设计及其流程图

从整体出发划分功能模块、安排程序结构并画出程序设计流程图。如根据系统的要求将程序大致分为:数据采集模块、数据处理模块、算法模块、串口接收发送模块、控制输出模块、故障诊断模块等,并规定每个模块的任务及其相互间的关系等。

程序流程图是用图形的方法将程序设计思路及程序流向完整地展现在平面图上,使程序结构直观、一目了然,有利于程序的审核、查错和修改。流程图通常在编写程序之前绘制,有时也在编写过程中绘制。画流程图时先画出简单的功能性流程图,然后对功能性流程图进行扩充和具体化,如存储空间的分配、寄存器的应用、标志位的说明等,进而再画出详细的流程图。先画主程序流程图,再画各模块的流程图。好的流程图可大量节省源程序的编辑、调试时间,保证程序的质量和正确性。也可以不画流程图,但应保证在设计者的头脑里有明确、清晰的编程思路和流程图。

在设计汇编程序时,还需要编制详细的资源分配表,包括 RAM 单元的分配、参数定义、位地址定义、使用的寄存器、数据指针的定义、各外围芯片的地址、子程序的功能等,以备程序的编制、检查和修改。

#### 4. 编写源程序

经上述几个步骤后进入编程阶段。根据系统总体设计的要求,按照流程图所设定的结构、算法和流向,选择合适的指令顺序编写,所编写的程序即为应用系统的源程序。在编写源程序时,要养成良好的注释习惯,即在每条指令后面或一段完整功能程序的开始和结尾部分说明该指令的功能,或说明程序完成的功能、入口参数、出口参数、具体的编程时间等,以备程序的检查和修改,增加程序的可读性、可维护性。

#### 5. 源程序的汇编与调试

把汇编语言源程序翻译成单片机能识别的机器语言(即目标代码)的过程称为汇编。所以汇编语言编写的源程序还需要汇编成目标代码。

汇编有人工汇编和机器汇编两种。人工汇编即用手工作逐条将汇编语言指令转换成机器码指令。人工汇编需要经过两次编译才能完成,第一次完成指令码的翻译,第二次完成地址偏移量的计算。人工汇编简单易行,但程序较长较复杂时效率较低、出错率高,在单片机发展的初期使用。随着计算机编程语言的丰富以及计算机的不断普及,出现了许多编译软件,能够利用计算机自动将汇编源程序(助记符形式)翻译成目标代码(机器码),称之为机器汇编。机器汇编实际是人工汇编的模拟。目前常用的编译软件有 Med Win、Keil C 等,许多开发系统、仿真器等都具有将汇编源程序编译成目标代码的功能。编译后的文件内容为十六进制码或二进制码格式,其文件名为 \*.hex 或 \*.bin 形式。机器汇编速度快、效率高、正确无误,已经取代了人工汇编。

对于大型的应用程序,编写完成即能成功运行的可能性较小,必须反复对程序进行严格、全面的调试和验证以及现场运行,直到完全正确、符合设计要求。

单片机应用程序的调试借助于相应型号的单片开发系统进行在线仿真调试,调试完成后利用编程器将程序下载到单片机中运行。具体仿真调试和下载参见本章 5.5 节。

#### 6. 系统软件的整体运行与测试

程序调试通过下载到单片机后,将整个系统硬件完整连接,进行系统的总体测试。测试一般由两个阶段组成,第一阶段由程序员根据功能要求自行测试,写出测试报告;第二阶段由专业程序测试人员完成,根据程序员的软件使用说明和功能进行测试,并出具测试结果和建议。

## 7. 总结归纳后进一步编写程序说明文件

程序说明文件是对程序设计工作进行的技术总结,有利于程序的后续修改、开发和经验交流,而且是正确使用、扩展和维护程序的必备文件。一般应包括以下几个方面的内容:

- 程序设计任务书,包括功能要求和技术指标;
- 程序流程图、RAM 分配表、参数定义、位地址定义、带注释的源程序清单等;
- 数学模型和应用的算法;
- 实际功能及技术指标测试结果说明书;
- 软件使用及维护说明书。

对于实际单片机应用系统的开发,有可能只包括其中的几个步骤,如一些系统可能不需要数学模型和算法,应根据具体情况确定软件的开发步骤。

## 5.2 源程序的基本格式及编辑环境

### 5.2.1 源程序的基本格式

AT89S 系列单片机汇编语言源程序的格式基本相同,但不同型号的单片机内部资源不同,如中断源个数不同、SFR 不同等,所以对应的源程序略有不同。编写源程序时应注意以下几个方面。

#### 1. AT89S52 的中断矢量分配

AT89S52 共有 8 个中断源,6 个矢量入口地址。其中断矢量入口地址分别为:0003H、000BH、0013H、001BH、0023H、002BH,另外,0003H~0032H 共 48 个单元被保留,专门用于中断服务程序。

#### 2. 程序的起始

对于 AT89S 系列单片机,没有程序启动运行指令,系统复位后立即启动并开始执行源程序。单片机复位后程序计数器指针 PC 值为 0000H,所以单片机复位后一定从程序存储器的 0000H 单元开始执行,而 0000H~0003H 之间的空间不足以存放主程序,因此,在 0000H~0003H 单元必须放置一条转移指令(AJMP 或 LJMP),转向主程序。由中断矢量分配表可知,0003H~0032H 为中断矢量区,应用系统主程序必须跳过这一区域,所以主程序起始地址一般设置在 0033H 之后。

#### 3. 中断服务程序

由中断矢量分配表可知,中断矢量区分配给每个中断源的中断服务程序的地址空间只有 8 个单元,一般是不够的,所以在中断矢量的入口处也放置一条转移指令,而实际的中断服务程序则安排在主程序地址空间之外的地址单元区段。

#### 4. 程序字节

程序字节(即程序长度)一定要小于等于程序的存储空间,AT89S52 具有 8 KB 的 Flash 程序存储器,所以编写程序的字节数一定要小于等于 8 KB,大于 8 KB 程序运行将出错。如程序量较大,可以选用具有较大 Flash 存储空间的单片机,如 AT89S53(12 KB Flash)或者外扩程序存储器 EPROM 等。

### 5. 伪指令

伪指令在汇编程序中具有一定的控制作用,其用法一定要遵循指令格式。

根据以上情况,源程序的基本格式及地址分配举例如下。

```

                                ORG    0000H
                                LJMP    MAIN      ;转向主程序
                                ORG    0003H
                                LJMP    INT0      ;转向外部中断 0 服务程序
                                ORG    000BH
                                LJMP    TIMER0    ;转向定时器 0 中断服务程序
                                ⋮
                                ORG    002BH
                                LJMP    TIMER2    ;转向定时器 2 中断服务程序
                                ORG    0040H
MAIN: SETB    IT0      ;主程序从 0040H 开始
      SETB    EX0      ;主程序初始化
      SETB    EA
      ⋮
      LCALL   DISP     ;调用显示子程序
      LCALL   DISPOSE   ;调用数据处理子程序
      ⋮
                                ORG    0300H
DISP:  ⋮                ;显示子程序
      ⋮
DISPOSE: ⋮             ;数据处理子程序
      ⋮
                                ORG    0400H
INT0:  ⋮                ;外部中断 0 中断服务程序
      ⋮
                                ORG    0500H
TIMER0: ⋮              ;定时器 0 中断服务程序
      ⋮
                                ORG    0600H
TIMER2: ⋮              ;定时器 2 中断服务程序
      ⋮
      ⋮                ;其他中断服务程序
                                ORG    0700H
TABDB: DB    12H,56H,3FH ;固定表格区段
      ⋮
      END              ;程序结束

```



主程序段一般包括内部 RAM 单元设置、中断设置、参数设置,以及外围芯片的地址、参数设置等初始化程序,各个模块子程序的调用、中断等待等。主程序是整个源程序的核心,一定要安排好各部分的先后顺序,保证程序的正常执行。

上述地址分配在实际应用中应视具体情况而定。

### 5.2.2 源程序的编辑环境

完成了系统软件的总体设计以后,就可以在 PC 上编写程序了。用于编写汇编程序的编写环境很广泛,可在许多文字编辑窗口进行输入,如在写字板、记事本,或专门的汇编编辑、调试软件,如 Med Win、Keil C 中进行,文件名以 .asm 为后缀。

因为编写完成的汇编软件还需要在特定的编译环境下进行语法编译,所以一般情况下,源程序直接在编译或仿真环境下进行编写。编写前首先在 PC 上安装需要的软件,如 Med Win 或 Keil C,并进行适当的配置,然后建立工程,就可以编写源程序了。

## 5.3 程序设计方法

汇编语言程序设计并不难,但要编写出质量高、可读性好、存储容量小和执行速度快的优秀程序并非很容易。要做到这一点,只掌握汇编语言的指令还远远不够,还需要掌握程序设计的基本方法,并且不断吸取、总结经验,提高编程技巧。汇编程序结构及基本设计方法有以下几种:

- 顺序结构程序设计;
- 分支结构程序设计;
- 循环结构程序设计;
- 子程序结构程序设计;
- 中断结构程序设计。

### 5.3.1 顺序结构程序

顺序结构程序指程序中没有使用转移类指令的程序段,是程序设计中最基本、最简单的编程结构,所以也称为简单结构或直线结构。顺序结构程序按照指令存储的位置,由低地址到高地址按顺序一条一条地执行。这类程序结构简单,易于阅读和理解。

**例 5-1** 编写 16 位数加 1 程序,16 位数存储在 30H、31H 单元,低位字节在前。

**解**

```
ORG      1000H
MOV      A,    30H    ;取 16 位数的低位字节    ;双字节、单周期
ADD      A,    #01H   ;低位字节加 1            ;双字节、单周期
MOV      30H,  A      ;送 30H 单元              ;双字节、单周期
MOV      A,    31H    ;取高位字节              ;双字节、单周期
ADDC     A,    #0      ;高位字节加进位位        ;双字节、单周期
MOV      31H,  A      ;高位字节送 31H          ;双字节、单周期
```

上述程序段完成了 16 位数加 1 的功能,程序顺序执行,共占用 12 个程序存储器单元,程序执行时间为 6 个机器周期。如果单片机使用 12 MHz 晶振,则每个机器周期为  $1\mu\text{s}$ ,程序执行时间  $6\mu\text{s}$ 。顺序结构设计虽然简单,但任何程序都离不开顺序结构程序。

### 5.3.2 分支结构程序

分支结构程序的特点是程序中含有转移指令,使程序具有判断和选择能力。由于转移指令分无条件转移和有条件转移,所以分支程序也分无条件分支和条件分支两类。无条件分支程序中含有无条件转移指令(如 SJMP、AJMP 等),比较简单。条件分支程序中含有条件转移指令,比较复杂。AT89S52 的分支程序设计主要就是正确运用累加器 A 判零条件转移、比较条件转移、减 1 条件转移和位控制条件转移等 4 类指令进行编程。

**例 5-2** 已知 40H(VAR)单元内有一自变量 X,按如下条件编写程序求 Y(FUN)的值,并存入 41H 单元。

$$Y = \begin{cases} 1 & X > 0 \\ 0 & X = 0 \\ -1 & X < 0 \end{cases}$$

**解** 这是三支归一的条件转移问题,有“先分支后赋值”和“先赋值后分支”两种编程方法,流程如图 5.3.1 所示。

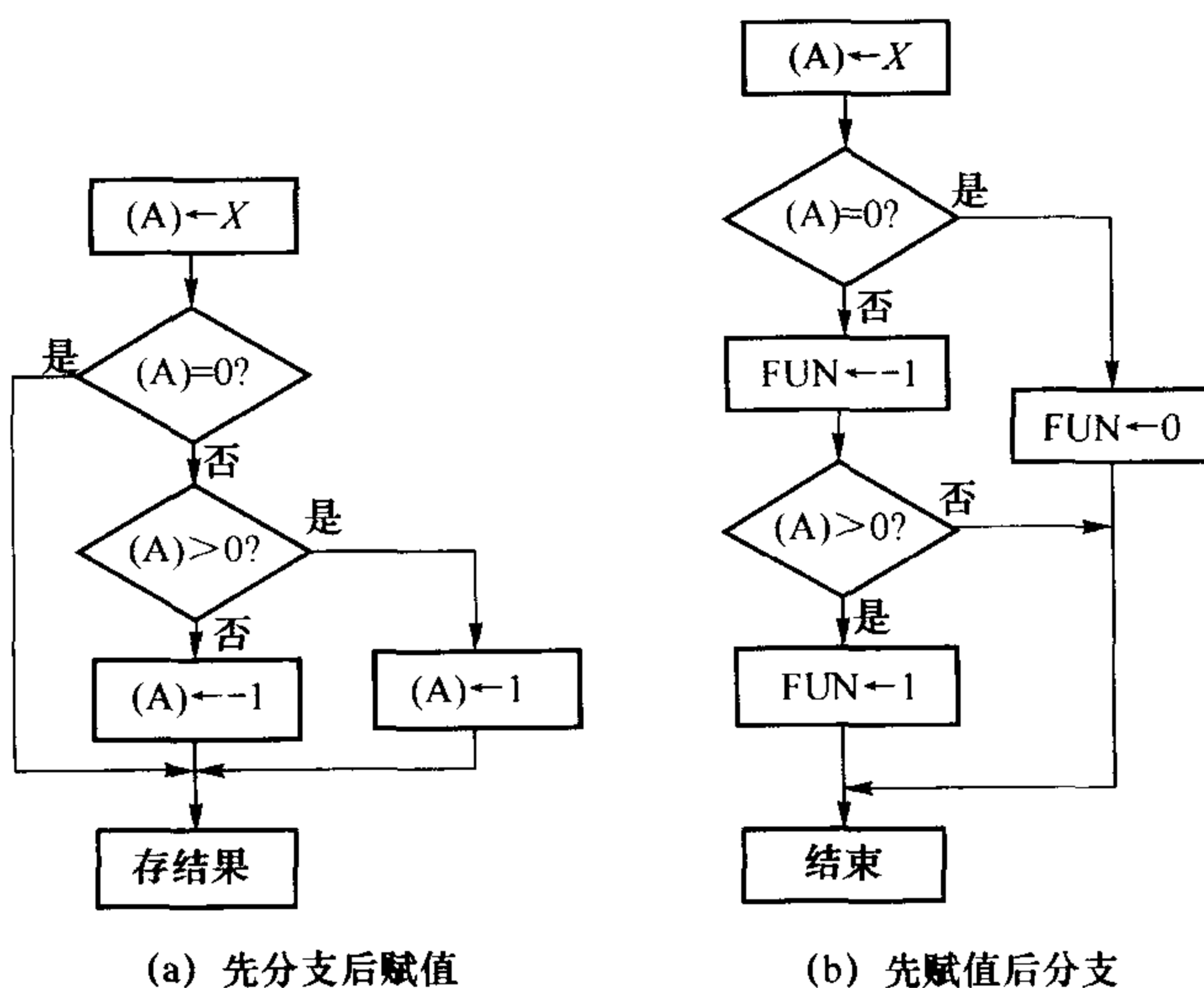


图 5.3.1 两种方案程序流程图

(1) 先分支后赋值程序。

```

ORG      2000H
VAR      DATA      40H
FUN      DATA      41H
XY:      MOV        A,      VAR      ;X→(A)
          JZ         DONE      ;X=0,转 ZERO
          JNB        ACC.7,    POS    ;X>0,转 POS
  
```

```

        MOV     A,          #0FFH      ;X<0, -1→(A)
        SJMP    DONE          ;转 DONE
POS:    MOV     A,          #01H      ;1→(A)
DONE:   MOV     FUN,        A          ;存 Y 值
        RET

```

(2) 先赋值后分支程序。把 VAR 内容送 A, 若 A=0, 则 FUN=0, 计算完成。若 A≠0, 则假设 A<0, 赋值 FUN=-1, 然后判断 A 是否小于 0, 若 A<0, 则计算完成; 若 A>0, 则重新赋值 FUN=1。

```

        ORG     2000H
        VAR     DATA     40H
        FUN     DATA     41H
XY:     MOV     A,          VAR        ;X→A
        JZ      ZERO       ;X=0, 转 ZERO
        MOV     FUN,        #0FFH     ;-1→FUN
        JB      ACC.7,     DONE       ;X<0, 转 DONE
        MOV     FUN,        #01H     ;1→FUN
        SJMP    DONE
ZERO:   MOV     FUN,        #00H     ;0→FUN
DONE:   RET

```

在汇编程序设计时, 有时会遇到多分支程序的设计, 根据运算结果在多个分支中选一, 并且分支转移的目标地址不是汇编或编程时确定的, 而是在程序运行时动态决定的。对于此类问题, 可以利用散转指令 JMP @A+DPTR 实现。

**例 5-3** 通过键盘扫描程序 KEYSCAN 读入键值 0~9 至累加器 A, 键值不同功能不同, 设计程序根据键值分别转入键控程序 KEY0~KEY9。即要求:

当(A)=0 时, 转键控程序 KEY0;

当(A)=1 时, 转键控程序 KEY1;

⋮

当(A)=9 时, 转键控程序 KEY9。

**解**

```

        ORG     1000H
        ACALL   KEYSCAN          ;读键值, 送累加器 A
        RL      A                 ;A 中内容乘 2
        MOV     DPTR, #JMPTBL    ;散转首址
        JMP     @A + DPTR        ;以 A 中内容为散转偏移量
        ⋮
JMPTBL: AJMP    KEY0             ;第 1 个键按下, 转向 KEY0 执行
        AJMP    KEY1             ;第 2 个键按下, 转向 KEY1 执行
        ⋮
        AJMP    KEY9             ;第 10 个键按下, 转向 KEY9 执行

```

```

KEY0:  : ;第 1 键程序处理段
KEY1:  : ;第 2 键程序处理段
      :
KEY9:  : ;第 10 键程序处理段

```

分支结构程序设计相比顺序结构程序设计难度要大,编写时最好先画出程序流程图,清楚程序的执行方向,然后再根据流程图进行程序的编写。

### 5.3.3 循环结构程序

循环结构就是多次重复执行某一程序段,直到满足结束条件,再向下顺序执行。该程序段通常称为循环体。循环结构并不减少程序执行时间,但可使程序紧凑,节省程序存储单元,增加可读性。循环次数越多,程序长度缩短越显著。循环结构通常由 4 部分组成。

#### 1. 循环初始化

循环初始化程序段位于循环程序开头,用于完成循环前的准备工作。一般需要设置循环控制计数器、数据指针、各工作寄存器及其他变量的初值等。

#### 2. 循环处理

这部分程序位于循环体内,需要重复执行,是循环结构的核心,程序编写应尽量简练,以缩短程序的执行速度。

#### 3. 循环控制

循环控制也在程序体内,用于控制循环执行次数。不断修改循环计数器、数据指针,判断循环结束条件是否满足。当循环次数已到时,则跳出循环控制,顺序执行后续程序。

#### 4. 循环结束

处理、存放循环程序的执行结果以及恢复各工作单元循环前的初值。循环程序有两种编制方法,具体实现如图 5.3.2 所示。

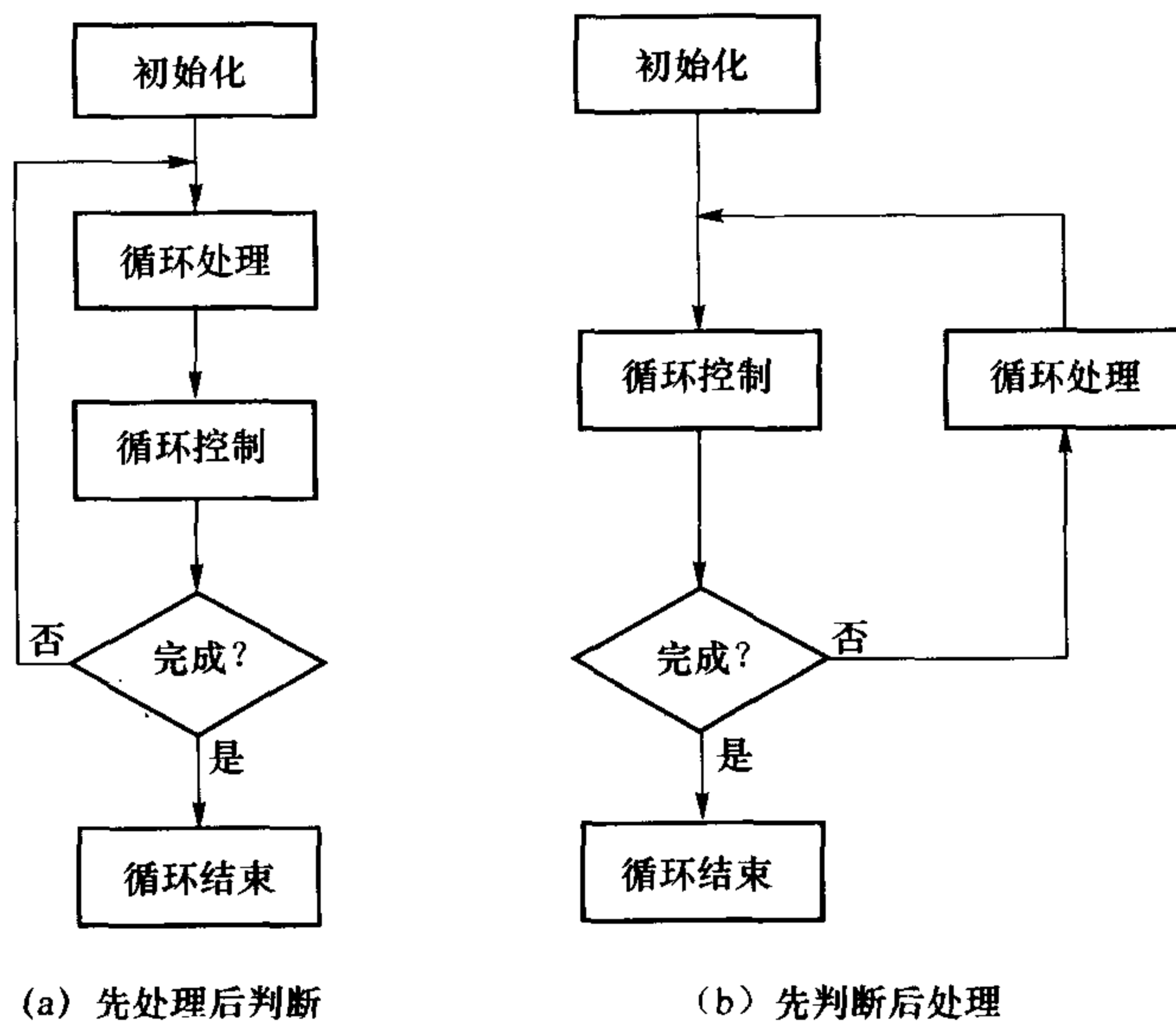


图 5.3.2 循环程序结构类型

**例 5-4** 已知单片机内部 RAM 的 BLOCK 单元开始有一无符号数据块,数据块长度在 LEN 单元,编写程序,求数据块中数据的累加和并将其存入 SUM1 和 SUM2 单元。

**解** 为了全面了解两种循环结构,现编写两种方案的程序。流程如图 5.3.3 所示。

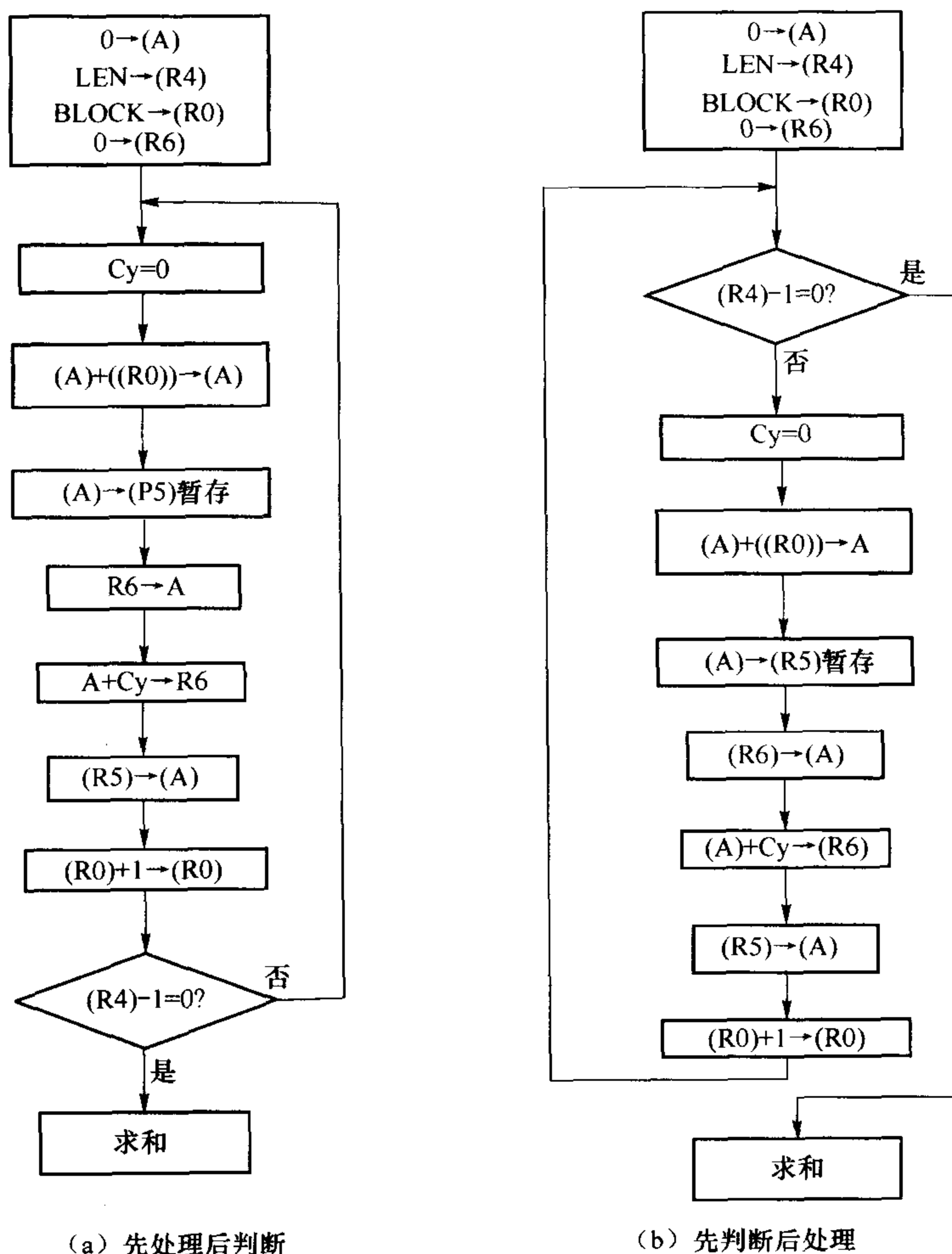


图 5.3.3 两种方案的程序流程图

#### (1) 先处理后判断程序

	ORG	1000H	
LEN	DATA	30H	
SUML	DATA	31H	;累加和低 8 位
SUMH	DATA	32H	;累加和高 8 位
BLOCK	DATA	50H	
	CLR	A	;A 清 0
	MOV	R6, #0	;R6 清 0, 暂存累加和高 8 位
	MOV	R4, LEN	;块长度送 R4
	MOV	R0, #BLOCK	;块起始地址送 R0

```

LOOP:  CLR    C                ;清进位位 Cy
        ADD    A,@R0           ;(A) + ((R0))送 A
        MOV    R5,A            ;转存累加器 A 值
        MOV    A,R6            ;取累加和高 8 位
        ADDC   A,#0H           ;计算高 8 位
        MOV    R6,A            ;累加和高 8 位送 R6
        MOV    A,R5            ;取累加和低 8 位
        INC    R0              ;修改数据块指针
        DJNZ   R4,LOOP         ;循环次数未到,转 LOOP
        MOV    SUML,A          ;循环次数到,存累加和低 8 位
        MOV    SUMH,R6         ;存累加和高 8 位
        END

```

## (2) 先判断后处理程序

```

                ORG    1000H
LEN DATA 30H
SUML DATA 31H                ;累加和低 8 位
SUMH DATA 32H                ;累加和高 8 位
BLOCK DATA 50H
        CLR    A                ;A 清 0
        MOV    R6,#0            ;R6 清 0,暂存累加和高 8 位
        MOV    R4,LEN           ;块长度送 R4
        MOV    R0,#BLOCK        ;块起始地址送 R0
        INC    R4               ;块长度加 1,因先判断后处理,执行减 1
        SJMP   CHECK           ;跳至循环控制
LOOP:  CLR    C                ;清进位位 Cy
        ADD    A,@R0           ;(A) + ((R0))送 A
        MOV    R5,A            ;转存累加器 A 值
        MOV    A,R6            ;取累加和高 8 位
        ADDC   A,#0H           ;计算高 8 位
        MOV    R6,A            ;累加和高 8 位送 R6
        MOV    A,R5            ;取累加和低 8 位
        INC    R0              ;修改数据块指针
CHECK: DJNZ   R4,LOOP         ;循环次数未到,转 LOOP
        MOV    SUML,A          ;循环次数到,存累加和低 8 位
        MOV    SUMH,R6         ;存累加和高 8 位
        END

```

上述两段程序是有区别的。若块长  $LEN \neq 0$ , 则执行结果相同。若块长  $LEN = 0$ , 则先处理后判断程序结果错误, 就是说, 先处理后判断程序至少有一次执行循环体内的程

序。所以在编写此类程序时,应注意采用合适的处理方法。

**例 5-5** AT89S52 的晶振为 12 MHz,设计 10 ms 延时程序。

**解** 12 MHz 晶振的机器周期为  $1\ \mu\text{s}$ ,可用双重循环写出延时程序。根据指令的执行周期计算延时时间。

```

                ORG      1000H
DEL:            MOV      R7, #40          ;单周期,1  $\mu\text{s}$ 
DEL1:           MOV      R6, #125         ;单周期,1  $\mu\text{s}$ 
DEL2:           DJNZ     R6, DEL2          ;双周期,2  $\mu\text{s}$ ,  $125 \times 2 = 250\ \mu\text{s}$ 
                DJNZ     R7, DEL1         ;双周期,2  $\mu\text{s}$ ,  $0.25 \times 40 = 10\ \text{ms}$ 
                RET

```

以上延时程序的延时时间计算不够精确,精确的延时时间计算如下。

$(1+125 \times 2) \times 40 + 1 = 10.041\ \text{ms}$ ,要想实现精确定时,可将程序修改如下:

```

                ORG      1000H
DEL:            MOV      R7, #40          ;单周期,1  $\mu\text{s}$ 
DEL1:           MOV      R6, #123         ;单周期,1  $\mu\text{s}$ 
                NOP                      ;单周期,1  $\mu\text{s}$ 
DEL2:           DJNZ     R6, DEL2          ;双周期,2  $\mu\text{s}$ ,  $123 \times 2 + 2 = 248\ \mu\text{s}$ 
                DJNZ     R7, DEL1         ;双周期,2  $\mu\text{s}$ ,  $(248 + 2) \times 40 + 1 = 10.001\ \text{ms}$ 
                RET

```

该段程序的延时时间为 10.001 ms。注意:使用软件延时过程中不允许中断,否则会严重影响延时时间的准确性。

### 5.3.4 子程序结构程序

子程序是一种能完成某确定任务的程序段,其资源需要为所有调用程序共享,因此,子程序在结构上应具有通用性和独立性。子程序的编写可以简化程序的逻辑结构,缩短程序长度,使程序调试变得轻松。

#### 1. 子程序编写时应注意的问题

##### (1) 保护现场和恢复现场

如果子程序使用主程序已经用过的寄存器和工作单元,子程序的运行可能会改变这些寄存器和工作单元的内容,从子程序返回,主程序再使用它们时就会发生错误。为避免这些错误,必须在子程序执行之前,将其内容保存起来,通常是进行压栈操作或改变工作寄存器的物理地址(由 RS0、RS1 决定),称为保护现场。子程序执行完成返回主程序后,再将压栈的内容取出,即弹栈或恢复原来工作寄存器的物理地址,并送数据回到原来的寄存器或单元,称为恢复现场。

保护现场和恢复现场有两种方式。



### ① 调用前保护调用后恢复

主程序中编写保护和恢复程序。调用子程序之前将某些参数转存或压栈,返回主程序之后将压栈的参数弹出。适用于每次调用需保护内容不同的情况。

### ② 调用后保护返回前恢复

子程序中编写保护和恢复程序。保护操作在子程序开始处编写,恢复操作在子程序的 RET 指令前编写。适用于保护内容固定的情况。

被保护的内容主要由子程序使用的寄存器确定。

## (2) 主程序和子程序之间的参数传递

主程序和子程序之间传递的参数主要有入口参数和出口参数。入口参数是指子程序需要的原始数据,由主程序传送给子程序;出口参数是由子程序根据入口参数执行子程序后获得的结果参数,由子程序传递给主程序。

参数的传递方法主要有以下几种。

### ① 用寄存器或累加器传递参数

将入口参数或出口参数放在工作寄存器或累加器中,程序最简单,运算速度也最快。缺点是工作寄存器数量有限,不能传递太多的数据;主程序必须先把数据送到工作寄存器;参数个数固定,不能由主程序任意设定。

### ② 利用寄存器传递参数地址

CPU 在主程序中把子程序入口参数地址利用寄存器 R0~R7 传递给子程序,可以大大节省传递数据的工作量并实现可变长度运算。如参数在内部 RAM 中,可用 R0、R1 传递;参数在外部 RAM 或程序存储器中,可用 DPTR 作地址传送。可变长度运算时,可用一个寄存器来指出数据长度,或者在 RAM 中使用结束标志,如特殊的 ASCII 码字符 0DH(CR,回车符)等。子程序执行完成后的出口参数也如此传递给主程序。

### ③ 利用堆栈传递参数

由于堆栈符合先进后出的原则,有一定规律可循,因此可以利用堆栈传递参数。主程序传递参数给子程序:主程序将参数依次压栈,子程序再依次弹栈并将其应用于子程序操作和运算。子程序传递参数给主程序:子程序将参数依次压栈,主程序再依次弹栈则可将参数应用于主程序。这种方法的特点是简单,能传递大量参数,不必为特定的参数分配存储单元,使用堆栈可以使中断响应的现场保护大大简化,但要注意堆栈指针的深度。

### ④ 利用位地址传递参数

当子程序的入口参数是字节中的某些位时,将这些位地址作为参数传递即可。

以上是子程序编写时要注意的问题。那么如何编写一段功能完整、可读性强、通用性好的子程序呢?在具体编写时还应遵循以下几个原则。

## 2. 子程序编写的原则

### (1) 子程序的第一条语句前必须有标号

子程序第一条指令的地址称为子程序的入口地址,第一条指令前必须有标号,因主程序调用子程序时要利用标号。该标号应以子程序的功能命名,一般用英文或汉语拼音表示,以便阅读时一目了然,如多字节加法运算可以用 ADDN 标记,延时程序可以用

DELAY标记等。

### (2) 标明子程序的入口参数和出口参数

子程序的入口和出口参数在子程序所加的注释中标明,一般附在子程序的开头,先说明子程序的功能,再说明入口参数和出口参数保存的位置。目的是增加程序的通用性和可读性,方便调用和调试。

### (3) 子程序的结尾必须有 RET 指令

子程序执行完成后一定要返回主程序,而子程序是通过 RET 指令返回到主程序的,所以在子程序的结尾必须安排 RET 指令,以便子程序顺利返回主程序。

### (4) 子程序中使用相对转移指令

为使所编写的子程序可以放在 64 KB 程序存储器的任何地方,并保证能被主程序正常调用,在编写子程序时必须使用相对转移指令而不使用其他转移指令,以便汇编时生成浮动代码。

### (5) 注意堆栈区要满足要求

子程序在调用和返回时,隐含两次堆栈(压栈和弹栈)操作,并且有可能在子程序中再调用子程序(即子程序嵌套),会导致多次堆栈操作,为了保证堆栈能正常进行,编程时应注意堆栈区要满足要求,或根据堆栈区安排子程序嵌套深度。

### (6) 保护现场和恢复现场

参见本小节中关于保护现场和恢复现场的内容。

## 3. 子程序编写举例

**例 5-6** 编写存取共阴极数码管对应的显示代码子程序。

**解**

```

;程序名称:SEGDISC
;功能:取得共阴极数码管对应显示代码程序
;入口参数:(R3) = 要显示的数字,如 0、1 等
;出口参数:(R4) = 显示数字的代码

                ORG          1000H
SEGDISC: PUSH ACC                        ;ACC 压栈,保护现场
                MOV          A,          R3      ;取显示内容
                MOV          DPTR,      # SEGTAB ;显示码首地址
                MOVC          A,          @A + DPTR ;查表得对应显示码
                MOV          R4,          A      ;显示码送 R4
                POP          ACC                ;累加器 ACC 弹栈,恢复现场
                RET                                ;子程序返回

                ORG          8000H
SEGTAB:  DB 3FH, 06H, 5BH, 4FH, 66H      ;对应于字符 0,1,2,3,4
          DB 6DH, 7DH, 07H, 7FH, 67H      ;对应于字符 5,6,7,8,9

```

其中,SEGTAB 不属于子程序的内容,可以放于程序的任何位置。

该程序使用了寄存器 R3、R4 进行参数传递。因子程序中已经设计了保护现场和恢

复现场的操作,所以主程序中可直接调用该子程序,不需要再进行现场的保护和恢复操作。主程序可如下编写:

```

                ORG      0000H
                LJMP     MAIN
                ORG      0050H
MAIN:          MOV      SP, #70H          ;设置堆栈指针
                ACALL    SEGDISC          ;调子程序
                ⋮
                END

```

#### 例 5-7 编写多字节无符号数相加子程序。

解

```

;程序名称:ADDN
;功能:多字节无符号数相加程序
;入口参数:(R0)=被加数低位地址指针
;          (R1)=加数低位地址指针
;          (R2)=字节数
;出口参数:(R0)=和数高位地址指针
                ORG      0800H
ADDN :         CLR      C                ;清进位 Cy
ADDN1:         MOV      A,@R0            ;被加数送 A
                ADDC     A,@R1           ;被加数与加数执行带进位加法
                MOV      @R0,A           ;和送 R0 指向的指针
                INC      R0              ;修改被加数指针
                INC      R1              ;修改加数指针
                DJNZ     R2, ADDN1        ;判断字节数加完否
                JNC      DONE            ;无进位,结束
                MOV      @R0,#01H        ;有进位,存入 01
                RET                     ;子程序返回
DONE:         DEC      R0                ;R0 内容减 1,因上面程序执行了
;          ;一次加 1 指令
                RET                     ;子程序返回

```

本程序的参数传递是通过寄存器传递参数地址完成的。程序执行后,被加数被冲掉。调用该程序的主程序可如下编写:

```

                ORG      0000H
                LJMP     MAIN
                ORG      0050H
MAIN:          MOV      SP, #70H          ;设置堆栈指针
                MOV      00H,C           ;进位位转存 20H.0

```

```

PUSH    20H                ;20H RAM 单元入栈,保护现场
MOV     R0,#30H            ;被加数地址指针
MOV     R1,#50H            ;加数地址指针
MOV     R2,#10             ;相加字节数:10
ACALL   ADDN               ;调用子程序
POP     20H                ;弹栈 20H,恢复现场
      ∴
END

```

**例 5-8** 编制程序将 RAM 单元 VAR1、VAR2 中的双字节十六进制数转换成 4 位 ASCII 码,存放于 R1 指向的 4 个内部 RAM 单元。

**解** 由于 VAR1、VAR2 中高、低 4 个数需要转换成 ASCII 码,所以将 ASCII 码转换程序编写成子程序,以便重复调用。子程序如下:

```

      ;程序名称:HEX2ASC
      ;功能:十六进制数到 ASCII 转换
      ;入口参数:堆栈
      ;出口参数:堆栈
ORG    0600H
HEX2ASC: MOV    R0,SP        ;取堆栈指针
      DEC     R0
      DEC     R0            ;堆栈指针减 2
      XCH     A,@R0        ;A 内容压栈保护,取出 HEX 内容送 A
      ANL     A,#0FH        ;取 A 中的低 4 位
      ADD     A,#02H        ;A 中内容加 2,得偏移量
      MOVC    A,@A+PC       ;查表
      XCH     A,@R0        ;将 ASCII 码压栈,A 中内容恢复
      ;单字节指令
      RET          ;子程序返回,单字节指令
      DB      '0123456789'  ;十六进制数的 ASCII 字符表
      DB      'ABCDEF'

```

说明 1:本程序通过堆栈传递参数。

说明 2:堆栈指令减 2 的原因是主程序调用子程序时隐含两次压栈,所以调用子程序后,栈顶指针变成了 SP+2,要取出原来压栈的内容必须减 2。

说明 3:A 中内容加 2 的原因是查表指令到表首地址有两条单字节指令。

主程序完成 VAR1、VAR2 中十六进制到 ASCII 码的转换,主程序编写如下:

```

ORG    0000H
LJMP   MAIN
ORG    0050H
VAR1    DATA    30H

```

	VAR2	DATA	31H	
MAIN:	MOV	SP, #60H		;设置堆栈指针
	MOV	A, VAR1		;VAR1 内容送 A
	SWAP	ACC		;交换 A 高低字节,先求高位的 ASCII 码
	PUSH	ACC		;压栈
	ACALL	HEX2ASC		;HEX 转成 ASCII 码子程序
	POP	ACC		;取出 ASCII 码
	MOV	@R1, A		;存 ASCII 码
	INC	R1		;修改指针
	PUSH	50H		;求低位的 ASCII 码
	ACALL	HEX2ASC		
	POP	ACC		
	MOV	@R1, A		
	INC	R1		
	MOV	A, 51H		
	SWAP	A		
	PUSH	ACC		
	ACALL	HEX2ASC		
	POP	ACC		
	MOV	@R1, A		
	INC	R1		
	PUSH	51H		
	ACALL	HEX2ASC		
	POP	ACC		
	MOV	@R1, A		
	:			
	END			

### 5.3.5 中断服务程序

中断服务程序对实时事件请求作必要的处理,使系统能实时地并行完成各个操作,中断服务程序必须包括现场保护、中断服务、现场恢复、中断返回 4 个部分。中断服务程序编写方法与子程序类似,同时应注意以下问题。

(1) 在中断程序的结尾一定要使用 RETI,以便返回到主程序调用处。

(2) 中断服务程序中要清除中断标志,以免重复进入。具体标志和清除方法参见各中断部分。

(3) 中断服务程序的长度应尽量短小,以免执行时占用 CPU 过多时间。所以主程序与中断服务程序之间的数据交换多采用标志位。

(4) 中断嵌套深度受堆栈区的影响。系统复位后,栈指针 SP 的初始值为 07H,与工

作寄存器区重叠,所以程序中一般要重新定义。AT89S52 内部虽有 256 B 的 RAM,但堆栈区需利用低 128 B 开辟,所以其堆栈深度有限。

## 5.4 C51 基础

C 语言是一种编译型程序设计语言,它兼顾了多种高级语言的特点,并具备汇编语言的功能。C 语言有功能丰富的库函数,运算速度快,编译效率高,有良好的可移植性,而且可以直接实现对系统硬件的控制。C 语言是一种结构化设计语言,支持由顶向下的结构化程序设计技术。C 语言的模块化程序结构可以使程序模块实现共享。在 C 语言的可读性方面更容易借鉴前人的开发经验,提高程序的开发水平。

C 语言应用于单片机编程除了上述特点外,还有以下突出特点:编译器可以自动完成变量存储单元的分配,省去了分配和记录存储单元的烦琐;不必对单片机和硬件接口的结构有很深入的了解,省去了单片机的漫长学习时间;具有良好的可移植性,只要将程序略加改动就可以将其应用于其他类型的单片机,省去了更改单片机型号时重新编写程序的无奈和痛苦。因此利用 C 语言编写程序可以大大缩短目标系统软件的开发周期,程序的可读性明显增加,便于改进、扩充及研制规模更大、性能更完备的系统。C 语言的缺点是生成的目标代码较大,但随着大规模集成电路的飞速发展,片内 ROM 的空间做到 16/32 KB 的已经很多,所以代码较大已经不是重要的问题了。目前,支持硬断点的单片机仿真器已能很好地进行 C 语言程序调试,为使用 C 语言进行单片机编程提供了便利条件。因此,用 C 语言进行程序设计已成为单片机开发、应用的必然趋势。

C51 指 8051 系列的 C 语言编译器。本书的 C51 程序针对特定的编译器 Keil C51  $\mu$  Vision。

### 5.4.1 C51 的程序结构及编译环境

#### 1. C51 的程序结构

C51 的程序结构与一般的 C 语言程序基本相同。C51 程序是一个函数的集合,这个集合有且仅有一个名为 main 的函数,main 函数也称为主函数,是程序的起点,也是程序的终点。不论 main 函数在程序的什么位置,程序总是从 main 函数开始执行,当 main 函数所有语句执行完成后,程序执行结束。

函数相当于汇编程序的子程序。由于每个子程序是一个模块,所以 C51 程序设计称为结构化的程序设计。

C51 程序举例:

```
#include <at89s52.h>          /* 预处理伪指令 */
:
#define uint unsigned int      /* 定义伪指令 */
:
uchar rcv;                    /* 变量定义 */
```

```

uchar data flag
    :
sbit flag1 = flag^1;           /* 位变量定义 */
    :
void delay();                  /* 全局函数定义 */
void delay()                    /* 函数 */
{
    uchar m;
    for (m = 0; m < 1000; m++){};
}
void int0(void) interrupt 0 using 1 /* 中断服务程序,使用工作
                                     寄存器 1 区 */
{
    :
}
void main ()                    /* 主函数 */
{
    :
}

```

C51 程序基本由预处理器命令、变量定义、函数组成。利用 C 语言编程时,一般在程序的开始都要引入头文件。如 `#include <stdio.h>` 语句,说明该段程序包含了 `stdio.h` 头文件。C51 程序除了包含 C 语言标准的头文件外,一般还要包含与单片机硬件有关的头文件,如 `#include <at89s52.h>` 语句,说明该段程序包含 `at89s52.h` 头文件,与单片机有关的头文件中一般定义对应单片机累加器 A、片内 I/O 口、可位寻址单元的地址、中断矢量的入口地址以及特殊单元的名称定义等。包含头文件后,经头文件定义的 SFR 等可以在程序中直接用变量名称代替,如在头文件中定义

```
sfr TMOD      = 0x89 ;
```

程序中要给 TMOD 赋值时,直接利用 `TMOD=40` 就可以了,相当于汇编语言中的

```
MOV    TMOD,    #40
```

不同的单片机因其硬件结构不同,上述定义也有所区别,所以不同的单片机需要有自己的头文件。如 AT89C2051 与 AT89C51 属于同一个系列,但 AT89C2051 只有 P1 口和 P3 口,所以 AT89C51 的头文件与 AT89C2051 的头文件不同。而 AT89S52 与 AT89C52 相比,增加了双数据指针、辅助寄存器、看门狗控制寄存器、定时/计数器 2、中断源等,所以它们的头文件也不同,一般对于不同的单片机需定义不同的头文件。头文件可以在原来的基础上更改,以适应其他型号的单片机,如 `at89s52.h` 可在 `at89c52.h` 的基础上增加一些寄存器的定义即可。

C51 程序编写时注意以下几点。



(1) 每个 C51 程序都至少包含一个名为 main 的主函数,也可以包括一个 main 函数和若干其他函数。因此,函数是 C51 程序的基本单位。被调用的函数可以是编译器提供的库函数,也可以是用户根据需要自己编制设计的函数。

(2) C51 程序总是从 main 函数开始执行,不管 main 函数在程序中的什么位置。

(3) C51 程序中一般包含相关单片机硬件定义的头文件。

(4) 变量的定义要遵循 C51 变量定义原则。

(5) C51 程序书写格式自由,一行内可以写一条或几条语句。每条语句最后以分号结束。

(6) 可以用 /\* ... \*/ 对 C51 中的某段程序加注释,也可以利用 // 对某行内容加注释。

## 2. C51 的编译环境及开发过程

许多公司都开发了 C51 的集成开发环境(IDE)及开发系统,常用的 C51 集成开发环境有 Keil C51  $\mu$ Vision 和 Med Win 等软件。Keil C51  $\mu$ Vision 是 Keil Software, Inc/Keil Elektronik GmbH 开发的基于 51 内核的微处理器软件开发平台,内嵌多种符合当前工业标准的开发工具,可以完成工程建立和管理、编译、链接、目标代码的生成、软件仿真及硬件仿真等完整的开发流程。Med Win 是南京万利电子(南京)有限公司推出的配合其 Insight 仿真器的集成开发环境。

C51 源程序是 ASCII 文件,除了应用上述编译环境编写外,也可以采用如 EDIT、记事本、写字板等进行编写。C51 程序的开发过程如图 5.4.1 所示。

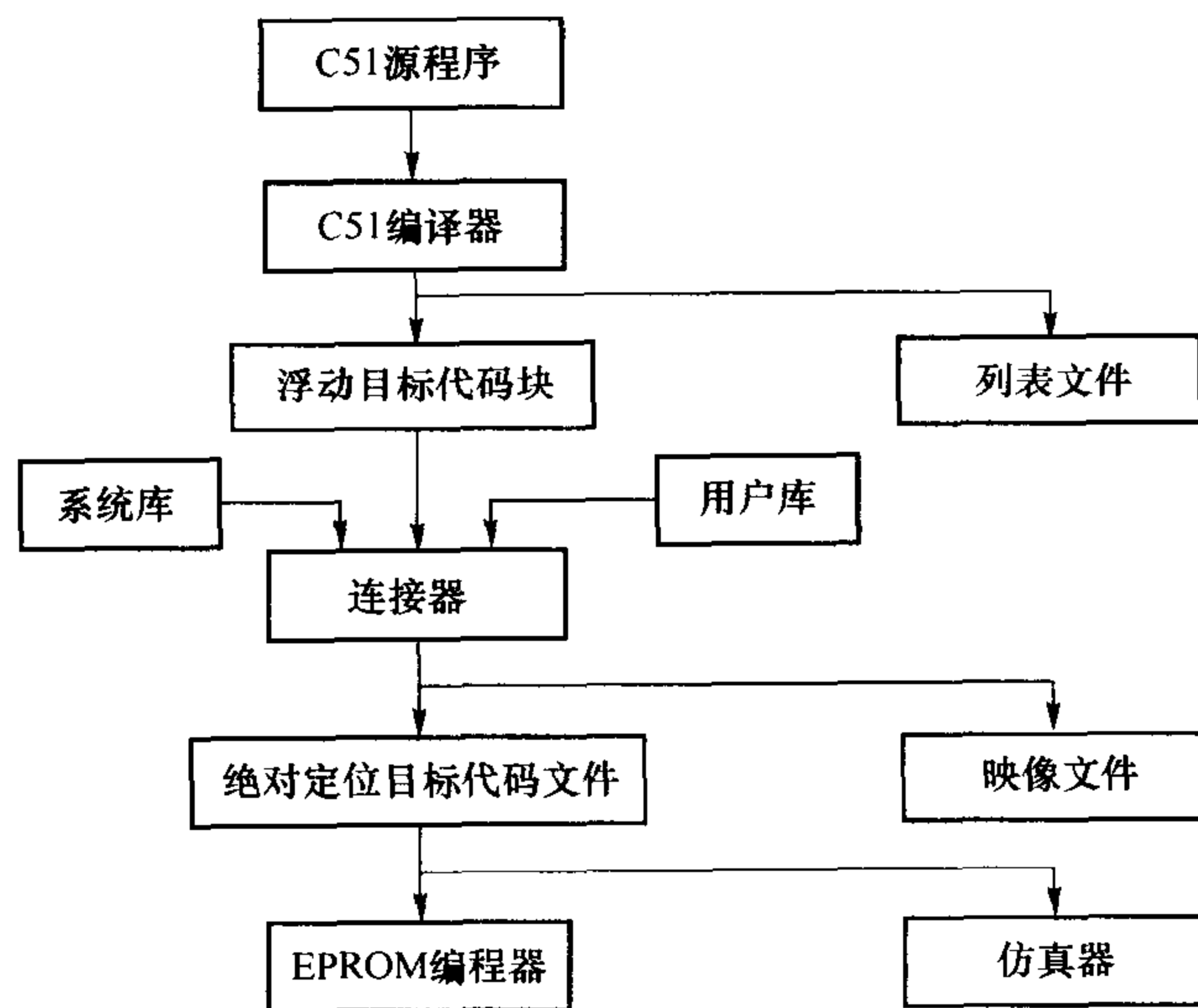


图 5.4.1 C51 程序开发过程

## 5.4.2 C51 的数据类型与存储类型

### 1. C51 的数据类型

C51 的数据类型与一般 C 语言相同。有以下几种数据类型:位型(bit)、无符号字符

型(unsigned char)、有符号字符型(signed char)、无符号整型(unsigned int)、有符号整型(signed int)、无符号长整型(unsigned long)、有符号长整型(signed long)、浮点型(float)和指针型。具体的数据类型、长度和值域如表 5.4.1 所示。

表 5.4.1 C51 的数据类型

数据类型	长度/bit	长度/B	值域范围
bit	1		0,1
unsigned char	8	1	0~255
signed char	8	1	-128~127
unsigned int	16	2	0~65 535
signed int	16	2	-32 768~32 767
unsigned long	32	4	0~4 294 967 295
signed long	32	4	-2 147 483 648~2 147 483 647
float	32	4	$\pm 1.176E-38 \sim \pm 3.4E+38$ (6 位数字)
double	64	8	$\pm 1.176E-38 \sim \pm 3.4E+38$ (10 位数字)
一般指针	24	3	存储空间 0~65 535

当数值计算结果隐含着另外一种数据类型时,数据类型可以自动进行转换。例如,将一个位变量赋给一个整型变量时,位型值自动转换成整型值,有符号的变量的符号也能自动进行处理。这些转换也可以用 C 语言的标准指令进行人工转换。

上述数据类型中只有 bit 和 unsigned char 两种数据类型可以直接转换成机器指令。如果不进行负数运算,编程时最好使用无符号格式(unsigned),以保证程序的运算速度并减少存储空间。有符号字符变量虽然只占一个字节,但需要进行额外的操作来测试代码的符号位,这会降低程序的代码效率。所以在 C51 编程时,应尽量避免使用大量的、不必要的数据类型,以减轻程序的代码,提高执行速度。

C51 中表示十六进制数据时,例如用 0x34 表示,与汇编语言中表示的十六进制数的 34H 等价。

为了书写方便,编程时常使用简化的缩写形式定义数据类型。具体方法是在程序开头使用预处理命令 #define。如:

```
#define uchar unsigned char
#define uint unsigned int
```

这样,在以后的编程中,就可以用 uchar 代替 unsigned char,用 uint 代替 unsigned int 来定义变量。C51 编程中变量可以定义成以上数据类型。如:

```
uchar send_data, rec_data;
```

## 2. C51 数据的存储类型

因为 C51 是面向单片机及其硬件控制系统的开发工具,利用 C51 编写的程序最后要转换成机器码,并下载到单片机中运行,而单片机中数据的存储空间共有 4 类:片内程序存储器空间、片外程序存储器空间、片内数据存储器空间、片外数据存储器空间。在利用

汇编指令编写程序时指令本身就确定了数据的读写位置,如 MOVX 指令用来实现外部数据存储器的读写,所以不必再说明。而利用 C51 编写的程序需要在程序中说明数据的存储空间,这样最后生成的目标代码中的数据才能按要求存储。所以 C51 编写程序时数据的定义除了前面加上数据类型外,还需要额外增加数据存储空间的说明。这一点与 C 程序是有区别的。数据的存储类型定义了数据在单片机系统中的存储位置,所以在 C51 中将变量、常量定义成各种存储类型,目的是将它们定位在相应的存储空间。

根据单片机硬件结构的特点,C51 定义了 6 种存储类型: data、bdata、idata、pdata、xdata、code,这些存储类型与 51 单片机实际存储空间的对应关系如表 5.4.2 所示。

表 5.4.2 C51 存储类型与 51 单片机存储空间的对应关系

存储类型	与单片机存储空间的对应关系
data	直接寻址片内数据存储区,访问速度快
bdata	可位寻址片内数据存储区,允许位与字节混合访问
idata	间接寻址片内数据存储区,可访问片内全部 RAM 地址空间
pdata	分页寻址片外数据存储区,由 MOVX @R0 访问
xdata	片外数据存储区,由 MOVX @DPTR 访问
code	程序代码存储区,由 MOVC @A+DPTR 访问

当使用存储类型 data、bdata 和 idata 定义常量和变量时,C51 编译器会将它们定位在片内数据存储区中(片内 RAM),对于 AT89S52 而言,该存储区为 256 B。变量的存储类型举例如下:

```
unsigned char data var1;
bit bdata flag;
float idata a,b,c;
unsigned int pdata temp;
unsigned char xdata array1[10];
unsigned int code array2[12];
```

上述语句定义了变量 var1、flag、a、b、c、temp 和数组 array1、array2。无符号字符型变量 var1 的存储类型为 data,定位在内部 RAM 区;flag 位变量的存储类型为 bdata,定位在片内数据存储区的位寻址区;a、b、c 浮点变量的存储类型为 idata,定位在片内数据存储区,并只能用间接寻址的方法访问;temp 无符号整型变量的存储类型为 pdata,定位在片外数据存储区,用指令 MOVX @Ri 访问;无符号字符型一维数组变量 array1 的存储类型为 xdata,定位在片外数据存储区,占据 10 B;无符号整型一维数组 array2 的变量类型为 code,定位在程序存储区,由 MOVC @DPTR 访问。

访问片内数据存储器(data,bdata,idata)比访问片外数据存储器(xdata,pdata)相对要快,因此可将经常使用的变量置于片内 RAM,而将规模较大的或不常使用的数据置于片外 RAM 中。

定义变量时,有时会略去存储类型的定义,此时,编译器会自动选择默认的存储类型,而默认的存储类型由存储模式决定。

### 3. C51 的存储模式

C51 有 3 种存储模式 SMALL、COMPACT 和 LARGE, 存储模式决定了变量默认的存储类型、参数传递区和无明确存储类型的说明, 如表 5.4.3 所示。

表 5.4.3 存储模式及说明

存储模式	参数及局部变量传递区域	范 围	默认存储类型	特 点
SMALL	可直接寻址的片内存储区	128 B	data	访问方便, 所有对象(包括堆栈)都必须嵌入片内 RAM
COMPACT	分页片外存储区	256 B/页	pdata	通过 Ri 间接寻址, 堆栈位于片内 RAM
LARGE	片外存储区	64 KB	xdata	通过 DPTR 间接寻址, 效率较低, 数据指针不能对称操作

若声明 `char str1`, 在 SMALL 模式下, `str1` 被定位在 data 存储区; 在使用 COMPACT 存储模式下, 则 `str1` 被定位在 pdata 存储区中; 在使用 LARGE 存储模式下, `str1` 被定位在 xdata 存储区。

具体采用哪种存储模式可以在 C51 集成开发环境中选择。

### 5.4.3 AT89S52 结构的 C51 定义

C51 是基于单片机的高级编程语言, 因单片机内部有特殊功能寄存器、I/O 接口、可位寻址单元等, 它们都对应某些固定的地址, 如累加器 A 的地址为 0E0H, 为了直接访问它们并方便编程, 一些 C51 编译器提供了与标准 C 语言不兼容、只适用于单片机进行 C51 编程的关键字, 用来定义这些单元。一般将它们定义在头文件中, 也可以在编程过程中定义。C51 中引入了两个关键字 `sfr` 和 `sbit` 进行相应的定义。

#### 1. 关键字 `sfr`

语法: `sfr sfr_name = int constant;`

说明: `sfr_name` 必须是一个寄存器的名字, 等号后面是该寄存器的地址, 一般用十六进制表示, 必须是常数, 不允许带有运算符的表达式, 这个常数值必须在特殊功能寄存器地址范围内。

功能: 用于定义单片机中的特殊功能寄存器、片内 I/O 接口。

**例 5-9** 利用 `sfr` 定义 P0、P1、PSW、A、B、SP、DPL、DPH。

**解**

```
sfr P0 = 0x80;
sfr P1 = 0x90;
sfr PSW = 0xD0;
sfr A = 0xE0;
sfr B = 0xF0;
sfr SP = 0x81;
sfr DP0L = 0x82;
sfr DP0H = 0x83;
```

## 2. 关键字 sbit

语法: sbit sbit\_name = bit address;

说明: sbit\_name 必须是一个位地址的名字, 等号后面是该位的地址, 位地址必须位于单片机的片内可寻址单元。因为位地址的赋值方法有 3 种, 所以上述语法有 3 种实现方法。

(1) sbit sbit\_name = sfr\_name ^ int\_constant;

sfr\_name 必须是已定义的 SFR 或片内 I/O 接口的名字, int\_constant 是该位在 sfr\_name 中的位置, 数值范围为 0~7。这种定义方法类似于汇编中位定义的方法 PSW.0。

例如:

```
sfr    PSW = 0xD0;           /* 定义 PSW 地址为 0xD0 */
sbit   OV = PSW^2;          /* 定义 OV 位为 PSW.2 */
sbit   Cy = PSW^7;          /* 定义 Cy 位为 PSW.7 */
```

(2) sbit sbit\_name = int\_constant ^ int\_constant;

这种方法是以一个整常数作为基地址, 该值必须在 0x80~0xFF 之间, 并能被 8 整除。前一个 int\_constant 是地址, 后一个 int\_constant 是该位在该地址中的位置, 数值范围为 0~7。

例如:

```
sbit   OV = 0xD0^2;          /* 定义 OV 位地址为 0xD2 */
sbit   Cy = 0xD0^7;          /* 定义 Cy 位地址为 0xD7 */
```

(3) sbit sbit\_name = int\_constant;

这种方法将位的绝对地址赋给变量, int\_constant 是位地址, 地址必须位于 0x80~0xFF 之间。

例如:

```
sbit   OV = 0xD2;            /* 定义 OV 位地址为 0xD2 */
sbit   Cy = 0xD7;            /* 定义 Cy 位地址为 0xD7 */
```

因为第(1)种方法不需要记忆寄存器地址, 所以在实际编程中应用居多。为了通用, 寄存器、I/O 接口、特殊位的定义一般写在相应单片机的头文件中。头文件可以用记事本等编辑。开始部分先说明应用的单片机类型, 然后再开始定义。

**例 5-10** 编写 AT89S52 部分寄存器及位定义的头文件。

**解**

```
/* -----
AT89S52.H
----- */
/* -----
Byte Registers
----- */
sfr P0 = 0x80;
```

```

sfr SP = 0x81;
sfr DP0L = 0x82;
sfr DP0H = 0x83;
sfr DP1L = 0x84;
sfr DP1H = 0x85;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr ACC = 0xE0;
    :
/* -----
P0 Bit Registers
----- */

sbit P0_0 = 0x80;
sbit P0_1 = 0x81;
sbit P0_2 = 0x82;
sbit P0_3 = 0x83;
sbit P0_4 = 0x84;
sbit P0_5 = 0x85;
sbit P0_6 = 0x86;
sbit P0_7 = 0x87;
    :
/* -----
Interrupt Vectors:
Interrupt Address = (Number * 8) + 3
----- */

#define IE0_VECTOR 0 /* 0x03 外部中断 0 */
#define TF0_VECTOR 1 /* 0x0B 定时器 0 */
#define IE1_VECTOR 2 /* 0x13 外部中断 1 */
#define TF1_VECTOR 3 /* 0x1B 定时器 1 */
#define SIO_VECTOR 4 /* 0x23 串行口 */
#define TF2_VECTOR 5 /* 0x2B 定时器 2 */
#endif

```

程序中应用寄存器或其中某位时,其名称应与头文件中定义的名称相同,并且大小写相同。包含上述头文件的 C51 程序中,给 P0 或 P0 某位赋值时,要如下编写:

```

P0 = 0x7F;          /* P0 赋值 */
if (P0_0)           /* P0.0 = 1, 执行 display(); 否则执行 dispose(); */
    display();

```

```
else  
    dispose();
```

如果将 P0 小写编译时就会出现错误。

sbit 除了有上述特殊位地址定义外,在实际编程中还可以用来定义位。

例如:

```
unsigned char bdata flag;    /* 定义 flag 为 bdata 无符号字符型变量 */  
sbit flag1 = flag^1;        /* flag1 定义为 flag 的第 1 位 */  
sbit flag2 = flag^2;        /* flag2 定义为 flag 的第 2 位 */
```

如此定义的 flag1、flag2 则可以在程序中作为标志位使用。

#### 5.4.4 C51 和汇编语言的混合编程

汇编语言具有程序结构紧凑、占用存储空间小,实时性强、执行速度快,能直接管理和控制存储器及硬件接口的特点,故此 C 语言并不能完全替代汇编语言。单独应用汇编语言或 C51 语言进行编程时,都是应用同一种语言编程,程序应用不同的语言进行编程时,称为混合编程。

混合编程中主要涉及两种情况:C51 程序调用汇编程序和汇编程序调用 C51 程序。

由于 C 语言提供了丰富的库函数,具有很强的数据处理能力,编程中对 51 单片机的寄存器和存储器的分配均由编译器自动管理,而汇编程序短小精干、执行速度快,因而混合编程时通常主程序应用 C51 编写,与硬件有关的程序应用汇编语言编写,所以程序中涉及到 C51 程序调用汇编程序。C51 程序调用汇编程序主要有列情况。

- 外围设备的驱动程序用汇编语言编写,但主程序采用 C51 程序完成。如一些板卡的驱动程序一般用汇编语言编写。
- 较为复杂的程序需要采用结构性较强的 C51 语言开发,但是部分程序要求较高的处理速度而必须使用更精练的汇编语言编写。如高速数据采集中的 A/D 转换程序。
- 程序中的部分代码因时序要求严格而使用汇编语言编写。如串行接口芯片的读写。

在实际程序开发中,有些程序以汇编语言为主体,如果涉及到复杂的数学运算,往往需要借助于 C 语言工具所提供的库函数和强大的数据处理能力,这就要求汇编程序中调用 C51 程序。

因此,在单片机应用程序的开发过程中,有必要掌握 C51 和汇编语言的混合编程方法。

要实现混合编程,必须掌握两种编程语言之间函数名的转换规则、参数的传递和函数返回规则、C51 程序中调用汇编语言以及汇编程序中调用 C51 程序的方法。

##### 1. 函数名的转换

混合编程中,需要将欲调用的每种语言编写的程序用一段单独的程序表示,即 C51 中用函数表示,汇编中用子程序表示。互相调用时,函数名称需要变化,即函数名的转换。C51 与汇编中函数名的转换规则如表 5.4.4 所示。



表 5.4.4 函数名的转换

C51 中函数说明	汇编中符号名	解 释
void func(void)	FUNC	无参数传递或不含寄存器参数的函数名不作改变,转入目标文件中,名字只是简单地转为大写形式
void func(char)	_FUNC	带寄存器参数的函数名加入“_”字符前缀以示区别,它表明这类函数包含寄存器内的参数传递
void func(char) reentrant	_? FUNC	对于重入函数加上“_?”字符串前缀以示区别,它表明该函数包含栈内的参数传递

如使用 C51 编写的函数 void display(void) 为无返回值,无参数传递的函数,在汇编中调用时其名称应为:DISPLAY。

## 2. 参数传递和函数返回规则

混合编程的关键是参数和函数返回值的传递必须有完整的约定,否则程序中无法取到传递的参数。两种语言必须使用同一规则。

C51 参数传递的规则:所有参数以内部 RAM 的固定位置传递给程序,若传递位则必须位于内部可位寻址空间中,让调用和被调用程序的顺序和长度(字节/字/字符/整数)一致。内部 RAM 相同标示的块可共享,进行汇编程序调用前,调用程序在块中填入要传递的参数,调用时程序假定所需值已在块中。

### (1) 参数传递

参数传递可使用寄存器、固定存储器位置或使用堆栈。堆栈传递参数支持重入。因为 51 系列有限的内部 RAM,不能保证堆栈足够大,所以函数不能调用自己本身。

CPU 寄存器中最多传递 3 个参数。这种参数传递技术产生高效代码,可与汇编语言相媲美。参数传递的寄存器规则如表 5.4.5 所示。

表 5.4.5 参数传递的寄存器选择

参数类型	char	int	long, float	一般指针
第 1 个参数	R7	R6, R7	R4~R7	R1, R2, R3
第 2 个参数	R5	R4, R5	R4~R7	R1, R2, R3
第 3 个参数	R3	R2, R3	无	R1, R2, R3

如:func1(int a,unsigned char b,int \*c) 中第一个参数 a,通过 R6、R7 传递;第二个参数 b,通过 R5 传递;指针变量 c,通过 R1、R2、R3 传递。

参数传递段给出汇编子程序使用的固定存储区,其首地址通过名为“? 函数名? BYTE”的 PUBLIC 符号确定。当传递位值时,使用名为“? 函数名? BIT”的 PUBLIC 符号。所有传递的参数放在以首地址开始递增的存储区内。

### (2) 函数返回

函数的返回值放入 CPU 寄存器,如表 5.4.6 所示。

表 5.4.6 函数返回值对应的寄存器

返回值	寄存器	说 明
bit	C	进位标志
(unsigned) char	R7	保存在 R7 中
(unsigned) int	R6,R7	高位在 R6,低位在 R7
(unsigned) long	R4~R7	高位在 R4,低位在 R7
float	R4~R7	32 位 IEEE 格式,指数和符号位在 R7
指针	R1,R2,R3	R3 放存储器类型,高位在 R2,低位在 R1

C51 程序调用汇编子程序时,假定当前选择的寄存器组、累加器 A 和寄存器 B、DPTR、PSW 都已改变。

3. C51 中直接插入汇编指令方式

Keil C 编译器支持 C51 程序中直接插入汇编语言,也可以调用汇编语言编写的子程序。编程时一些与硬件有关的操作,一般在 C51 中直接嵌入汇编指令,解决这个问题有两种方法。

(1) 使用 asm 功能

当在某一行写入\_asm “字符串” 时,可以把双引号中的字符串按汇编语言看待,通常用于直接改变标志和寄存器的值或作一些高速处理,双引号中只能包含一条指令。

```
格式: _asm
      "Assembler Code Here";
如:void ad_convert(void)
    {_asm "MOV A, #00H";
     _asm "MOV DPTR, #7FFFH";
     _asm "MOVX A, @DPTR";
    }
```

(2) 使用“#pragma ASM”功能

如果嵌入的汇编语言包含多行,可以使用“#pragma ASM”识别程序段,并直接插入编译通过的汇编程序到 C51 源程序中。

```
格式: #pragma ASM
      Assembler Code Here
      #pragma ENDASM
```

例 5-11 编写程序从 P1.0 口输出方波。要求 Keil C 环境下 C51 程序中嵌入汇编程序段。

解 程序如下:

```
#include <reg52.h>

sbit P10 = P1^0;          /* 定义位变量 P10 */
void main(void)           /* 主函数 */
```

```

{
while(1){
    P10 = ! P10;           /* P1 口输出取反 */
    #pragma ASM           /* 汇编程序段开始 */
        MOV        R4, #18
        DJNZ       R4, $    /* 延时等待 */
    #pragma  ENDASM       /* 汇编程序段结束 */
    }
}                          /* 程序结束 */

```

注意在 Keil C 环境下,内嵌汇编时要将 SRC\_CONTROL 激活。激活的方法是:在 Project 窗口中包含汇编代码的 C 文件上单击鼠标右键,选择“Options for ...”,单击右边的“Generate Assembler SRC File”和“Assemble SRC File”,使复选框由灰色变成黑色(有效)状态。

#### 4. C51 中调用汇编子程序方式

Keil C 支持在 C51 程序中调用汇编语言子程序。可以直接编写汇编程序,也可以利用 C51 先编写 C51 程序,再利用 Keil 工具产生汇编程序,再进一步修改得到需要的汇编语言程序。

先用 C51 编写出函数的主体,然后用 SRC 控制指令编译产生 src 文件,进一步修改这个 src 文件并另存为 \*.a51 文件,就得到了所需要的汇编函数。该方法让编译器自动完成各种段的安排,提高了汇编程序的编写效率。如在许多程序中会调用软件延时程序,但 C51 的延时程序不如汇编程序能准确控制延时时间,所以在要求准确控制时,应用汇编语言编写程序。实现的具体步骤如下。

(1) 先建立一个工程,并用 C51 编写延时程序,名称为 delay.c,程序如下:

```

#define uchar unsigned char
#define uint unsigned int
void delay (uint x)
{
    uchar k;
    while (x-- > 0)
    {
        for (k = 0; k < 125; k++)
            {;}
    }
}

```

(2) 编译(build)生成的 delay.src 文件如下,并将其更改为 delay.a51 文件。

```

;. \delay.SRC generated from: . \delay.c
NAME      DELAY
? PR? _delay? DELAY      SEGMENT CODE

```

```

PUBLIC _delay
; #define uchar unsigned char
; #define uint unsigned int
; void delay (uint x)
    RSEG ?PR? _delay? DELAY
_delay:          ;C51 所写的函数产生的汇编代码从这里开始
USING 0          /* 寄存器 0 区,可以根据需要更改 */
; ----- Variable 'k' 041' assigned to Register 'R5' ----- /* 说明 R5 中内容
为变量 k,参考表 5.4.5。传递两个参数,第一个参数 x,第二个参数 k */
; ----- Variable 'x' 040' assigned to Register 'R6/ R7' ----- /* 说明 R6,R7
中内容为变量 x */

; SOURCE LINE # 3
; {
; SOURCE LINE # 4
? C0001:
;     uchar k;
;     while (x-- > 0)
; SOURCE LINE # 6
        MOV     A,R7
        DEC     R7
        MOV     R2,R6          /* 寄存器 0 区,所以 R6 指内部 RAM 06H 单元 */
        JNZ     ? C0007
        DEC     R6
? C0007:
        SETB    C
        SUBB    A, # 00H
        MOV     A,R2
        SUBB    A, # 00H
        JC      ? C0006
;
; SOURCE LINE # 7
;     for (k = 0; k < 125; k++)
; SOURCE LINE # 8
        CLR     A
        MOV     R5,A
? C0003:
        MOV     A,R5
        CLR     C

```

```

        SUBB    A, # 07DH
        JNC     ? C0001
    ;           {;}
           ; SOURCE LINE # 9
        INC     R5
        SJMP    ? C0003
    ;           }
           ; SOURCE LINE # 10
    ;
    ;}
           ; SOURCE LINE # 12
? C0006:
        RET
    ; END OF_delay
        END

```

(3) 在工程中编写主函数,程序如下:

```

#include < reg51.h >
#define uchar unsigned char
#define uint unsigned int
extern uint delay(uint x);    /* 说明 delay 函数为外部过程 */
void main()
{
    uint    delay_time;
    delay_time = 40;
    delay(delay_time);
}

```

(4) 将 delay. a51 文件加入工程,再次编译工程,到此已经得到汇编函数的主体,修改函数里面的汇编代码就可以得到所需要的汇编函数。

也可以直接在 Keil C 环境下编写汇编程序,如下述延时程序:

```

PUBLIC      _DELAYMA
DELAYP      SEGMENT CODE
RSEG        DELAYP
_DELAYMA:   NOP                /* 汇编子程序开始 */
DELAY:      MOV ACC, # 250;
DEL:        NOP
            NOP
            DJNZ ACC, DEL
            MOV A, R6

```

```

        JZ EXIT
        DJNZ R6, DELAY
EXIT:    RET
END

```

函数名 `_DELAYMA`, 在 `DELAYMA` 前面加“`_`”表示有参数传递, 否则无法调用。函数名可以小写。

### 5. 汇编程序调用 C51 函数方式

汇编程序调用 C51 函数的要求是: 符合符号转换规则; C51 函数构成一个单独的文件; 在汇编语言文件中, 用 `EXTRN` 说明 C51 函数为外部过程。

设上述 `delay.c` 程序为需要调用的 C51 函数, 汇编程序调用该函数时, 首先定义该函数为外部函数, 再根据表 5.4.5 规定的参数传递规则为相应寄存器赋值, 就可以实现在汇编程序中调用 C51 函数。

**例 5-12** 编写 P1.0 输出方波程序, 延时程序为 `delay.c`, 利用汇编程序调用 C51 子程序。

**解** 程序清单如下:

```

        EXTRN    CODE (_DELAY)          /* 声明外部函数 */
        ORG 0H
        LJMP     MAIN
        ORG      0050H
MAIN:    CPL      P1.0
        MOV      R6, #17                 /* R6、R7 传递参数 x */
        MOV      R7, #38                 /* x = 0x1126, R6 高位、R7 低位 */
        LCALL    _DELAY                  /* 调用 C51 函数 */
        SJMP     MAIN
        END

```

### 5.4.5 C51 程序设计举例

本节以前面的汇编程序为例介绍 C51 程序的设计。

**例 5-13** 利用 C51 编写例 5-3 的键盘散转程序。

**解** 程序清单如下:

```

#include <reg52.h>
#define uchar unsigned char
void keyjmp(uchar key_data); /* 定义全局函数 */
void key0_dispose(void);
void key1_dispose(void);
void key2_dispose(void);
void key0_dispose(void)
{

```

```

/* 键 0 处理内容 */
}
void key1_dispose(void)
{
/* 键 1 处理内容 */
}
void key2_dispose(void)
{
/* 键 2 处理内容 */
}
/* 键散转子程序,入口参数:键值 key_data,返回参数:无 */
void keyjmp(uchar key_data)
{
    uchar key_temp;
    key_temp = key_data;
    switch(key_data)
    {
        case 0: {key0_dispose();
                  break; } /* 键处理子程序 */
        case 1: {key1_dispose(); break;}
        case 2: {key2_dispose(); break;}
        default: break;
    }
}
void main(void) /* 主程序 */
{
    keyjmp(0);
}

```

**例 5-14** 利用 C51 编写例 5-6 的存取共阴极数码管对应显示代码函数。  
**解**

;程序名称:SEGDISC  
 ;功能:取得共阴极数码管对应显示代码程序  
 ;入口参数:要显示的数字如 0、1 等  
 ;出口参数:显示数字的代码

程序清单如下:

```

#include <reg52.h>
#define uchar unsigned char
uchar getdiscode(uchar dis_data); /* 定义全局函数 */
uchar code segtab[10] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,
0x67}; /* 定义显示代码在程序存储器的空间 */

```

```

uchar getdiscode(uchar dis_data)
{
    uchar temp_data,i;
    i = dis_data;
    temp_data = segtab[i];
    return(temp_data);
}

void main (void)
{uchar x,y;
    x = 8;
    y = getdiscode(x);
}

```

**例 5-15** 利用 C51 编写初始化程序。要求:定时器 1 模式 2,产生串口波特率,定时器 0 模式 1。

**解**

```

void initialize(void)
{
    PCON = 0;
    SCON = 0x50;      /* 串口允许接收 */
    TMOD = 0x21;      /* 定时器 1 模式 2,自动装载;定时器 0 模式 1,16 位 */
    TH1 = 0xFD;      /* 波特率 19200 */
    TL1 = 0xFD;
    TR1 = 1;          /* 启动定时器 1 */
    RI = 0;           /* 清串口中断标志 */
    TH0 = 0xDC;      /* - 9216/256 */
    TL0 = 00;        /* - 9216%256 */
    IE = 0x90;        /* 开总中断 */
    ET0 = 1;          /* 开定时器 0 中断 */
}

```

## 5.5 程序调试与下载运行

源程序编辑、输入完成后,则可以进入源程序的调试和最后下载阶段,如图 5.5.1 所示。由图中看出,源程序的调试下载可能会经过几个循环才能最终完成。

编译是指在计算机中利用专用的编译软件对源程序的语法进行检查的过程,如 Med Win 及 Keil C 等。仿真就是使用可控的手段来模仿真实的情况,是对目标系统而言。仿真调试也称在线仿真调试,指利用单片机开发系统对目标系统的单片机进行实时仿真调试、在线运行的过程,主要用于目标系统的硬件和软件调试,以检查软件和硬件的错误。目标系统指根据总体设计要求完成的硬件系统。程序下载指利用编程器或下载器



将调试通过的程序的目标代码写到单片机的 Flash 或程序存储器的过程。

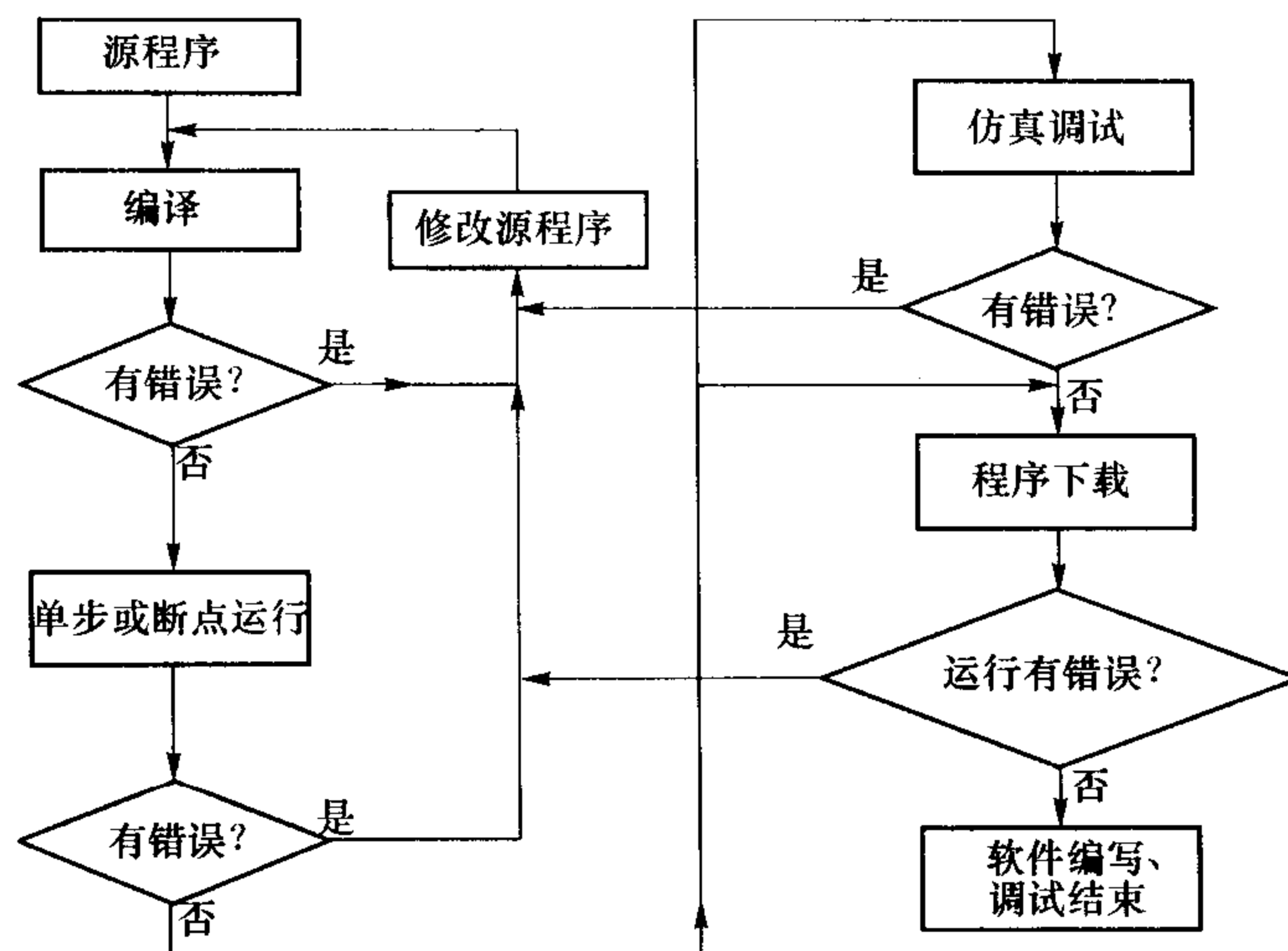


图 5.5.1 源程序的调试仿真流程图

### 5.5.1 单片机开发系统

单片机开发系统(装置)一般由计算机和仿真器组成,仿真器与计算机一般通过 RS232 串行接口、LPT 并行接口、USB 接口等相连,如图 5.5.2 所示。

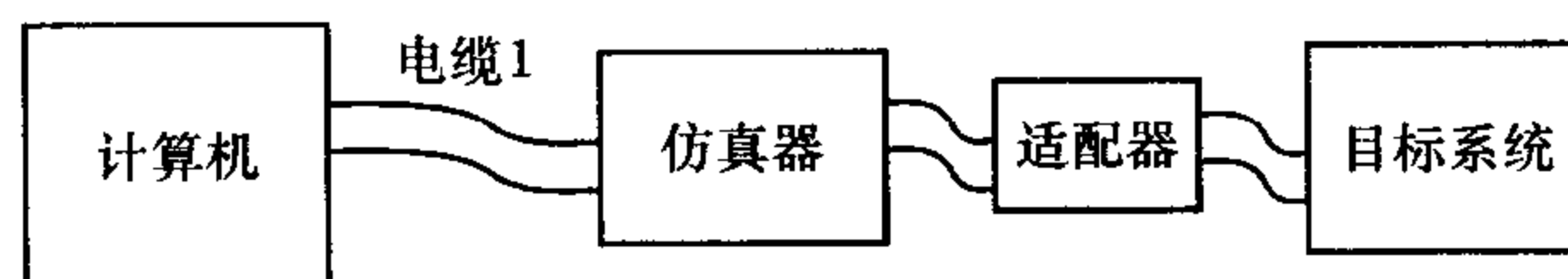


图 5.5.2 开发系统与目标系统连接图

图中电缆 1 是计算机与仿真器的连接电缆,根据仿真器与计算机的连接方式不同可以是并行、串行或 USB 电缆;仿真器通过适配器和目标系统相连。仿真器由仿真单片机、监控程序存储器、存放用户目标程序 and 数据的仿真存储器、存放断点的断点存储器、仿真单片机运行状态切换和读出修改数据的控制电路等组成,并提供和用户目标系统单片机插座相连的仿真适配器,仿真适配器要保证其上的插座与所仿真单片机的引脚、封装结构相同。

仿真器可分为通用型和专用型两类。通用型仿真器可以开发多种单片机,通过更换适配器即可进行多种型号的单片机仿真;专用型仿真器只能开发一种单片机。

单片机的开发系统是面向机器的,由于不同单片机系列的硬件组成结构、指令系统各不相同,其对应的开发系统也各不相同,因此在选用开发系统时要根据具体使用的单片机进行选择。虽然不同的单片机开发系统不同,但开发系统一般都应具有以下功能。

#### 1. 模拟仿真功能

能够实现程序的模拟仿真、调试,主要由开发系统支持的软件决定。

## 2. 在线仿真功能

开发系统中的在线仿真器应能仿真目标系统中的单片机,并能模拟目标系统的 ROM、RAM 和 I/O 接口,使在线仿真时目标系统的运行环境完全“逼真”于实际运行的环境,以实现系统的、完全的、一次性的仿真。

## 3. 调试功能

具有单步、硬件断点、连续、启停等运行方式,在任何运行方式下均能返回到监控状态,并支持高级语言的单步和断点运行。

能够读出和修改目标系统所有资源的状态,具有逻辑分析仪的功能,可以跟踪存储器信息的变化,并能显示某位的状态变化波形。

## 4. 辅助设计功能

能够支持汇编语言和高级语言,允许使用混合编程,并提供反汇编功能。

## 5. 用户操作界面

用户操作界面友好,方便操作,各种操作同时在集成环境下完成,并能实现编辑、编译、调试环境之间的快速切换。

# 5.5.2 源程序调试

源程序的调试必须在专用的调试环境下进行,如 Med Win 及 Keil C 等。源程序的调试一般包括两个过程:模拟调试和在线仿真调试。

## 1. 模拟调试

模拟调试指在 PC 的调试环境下模拟实际单片机运行的调试过程。用户只需要计算机以及编译环境,不需要单片机开发系统及硬件目标系统就可以对程序进行验证,因此调试与硬件无关的系统具有一定的优点,特别适合于偏重算法的程序仿真。模拟调试的缺点是无法完全仿真与硬件相关的部分,因此还要通过在线仿真调试来完成最终的设计。

模拟调试是源程序调试必须经过的过程。一般经过以下两个步骤。

### (1) 编译程序,检查汇编语言的语法是否正确

此过程检查源程序的语法是否正确,提示程序中错误的指令或格式,常称为编译。当编译出现错误时,系统会给出错误所在的行以及错误的提示信息。用户应根据这些提示信息,更正程序中出现的错误,反复编译直至完全正确为止。程序编译通过后,一般的编译软件在编译的同时输出对应的目标代码文件,该文件名格式为 \*.hex 或 \*.bin,这就是最后下载到程序存储器的十六进制或二进制文件。有些软件的输出文件格式事先需要设置,如 Keil C,请参照软件指导设置。

### (2) 模拟运行程序,观察程序的执行是否符合要求

程序检查没有错误后,进行单步或断点调试。目的是观察程序的流程是否正确,程序是否按要求执行,中断是否能正常进入等。因是模拟调试,所以有些在目标系统中需硬件确定的状态需要手动在 PC 上进行设置,如某个引脚的置位或清除、定时器中断标志的置位或清除等。如在检测能否正常进入定时器中断服务程序时,先将定时器中断服务程序的第一条指令设为断点,然后运行程序并观察程序能否进入断点执行,以及程序执行是否正确等。

## 2. 仿真调试

由于单片机本身没有自开发功能,必须借助于开发工具来排除目标系统样机中的硬件故障、程序错误,最后生成目标程序。模拟调试完成后,一般还要经过仿真调试。

仿真调试指使用单片机开发系统的单片机替代用户目标系统的单片机,并使其完成全部或大部分功能的调试过程。仿真调试必须配备相应的单片机开发系统。用户使用开发系统就可以对程序的运行进行控制,例如单步、断点、全速、查看资源等。仿真调试有利于发现硬件和软件的问题,简化程序的编写难度,缩短开发调试时间,所以在单片机应用系统的开发过程中占有非常重要的地位。

在实际的系统开发中,初级单片机学习者一般要经过上述两个过程(即进行模拟调试和仿真调试)才能最终使程序满足要求,而熟练的有经验的单片机开发人员一般不必进行仿真调试,如图 5.5.1 所示,程序模拟调试通过后直接将程序下载到单片机的 Flash 或 EPROM 中,然后运行目标系统,但这样做时一定要设置一些状态显示(如发光二极管、引脚的高低电平等),以检测程序运行是否正确。如果目标系统运行过程出现问题,如某些状态指示灯不亮、某些引脚输出错误或数据显示错误等,首先要根据具体的问题分清硬件和软件问题,对于硬件问题要核查电路设计是否合理、正确,印制板线路连接是否有短路、断路情况等,对于软件问题要重新修改程序。找到问题解决问题后,再反复经过上述过程调试,直至满足要求。

### 5.5.3 程序下载运行

程序调试通过后,进入到程序下载阶段。程序下载指的是利用编程器(烧写器或固化器)将调试正确的汇编程序的目标代码写入到 Flash 或 EPROM 中,以便单片机能够独立运行。对于 AT89S52 单片机,因其具有 ISP 在线编程功能,不用其他编程设备可以直接通过计算机串口,配合 ISP 下载软件对 AT89S52 进行在线编程。也可根据 3.4 节内容自行设计。另外,有许多厂家都生产编程器,如万利电子(南京)有限公司、广州周立功发展有限公司、南京西尔特电子有限公司等,按说明书进行操作即可将程序下载。选购编程器时可以从以下几个方面综合考虑。

#### 1. 需要编程的单片机类型

以目前需要编程的单片机为基准,并考虑将来的开发方向,有适当的冗余以备后续程序开发。有些开发系统同时具有编程器的功能。

#### 2. 与计算机的连接方式

编程器与计算机的连接方式一般有并行接口、串行接口、USB 接口连接方式。由于 USB 接口的即插即用的特点,目前有许多编程器通过 USB 接口与计算机相连,而且直接从 USB 接口取电,不需要再另加电源适配器,具有良好的应用前景。

#### 3. 售后服务情况

尽量选择售后服务好的公司的产品。

#### 4. 性能价格比

选购性能价格比高的产品,也可根据实际情况进行选购。

总之,源程序从编写到在单片机中能够正确运行,一般需要经过编译、仿真调试、下

载、独立运行几个过程。熟练掌握每个过程的操作对实际系统的开发和研制具有重要的作用。

## 习 题

1. 编写程序,求(30H)和(31H)单元内两数差的绝对值,结果保存在(40H)中。
2. 编写子程序,将(R0)和(R1)指出的内部 RAM 中的两个 3 字节无符号整数相加,结果送(R0)指出的内部 RAM 中。
3. 编写程序,将内部 RAM 中(50H~56H)的内容循环左移 4 位。
4. 编写多字节十进制数加法子程序。
5. 编写 4 位压缩 BCD 码转换成二进制整数的子程序。BCD 码高位地址指针为 R0,二进制整数结果存于 R3、R4 中。BCD 码放在 20H(高位)、21H 中,编写主程序并调用上述子程序,求对应的二进制整数。
6. 编写排序程序,将在内部 RAM 以 50H 为起始地址的单元中存放着的 10 个单字节无符号数,按从小到大的次序排列。
7. 编写比较两个 ASCII 字符串是否相等的程序。
8. 编写的应用程序为什么要调试? 开发单片机应用系统为什么要借助仿真器?
9. C51 编写程序有什么特点?
10. 编写延时 10 ms 的程序,晶振为 24 MHz。

## 实践训练

学完本章内容后,可进行集成环境的安装、调试,以及单片机应用系统的模拟、仿真调试,单片机晶振电路、复位电路的调试方面的实践训练。实践训练可按照下面给出的步骤进行。

1. 安装 Med Win 或 Keil C 集成开发环境。
2. 编写 AT89S52 的头文件,将其放于集成开发环境中。
3. 在 Med Win 或 Keil C 集成开发环境下建立工程文件,模拟调试书中程序,熟悉单步、断点调试,熟练查看 RAM、寄存器等的内容、反汇编码等。

4. 上机编写调试程序计算  $N$  个单字节数据的和,即  $Y = \sum_{K=1}^N X_K$ , 其中,  $N$  个数据  $X_N$  依次存放在 40H 单元开始的片内 RAM 区,结果  $Y$  放在 R3(和的高字节)、R4(和的低字节)和工作寄存器(2 区)中,并计算下列式子的结果:

(1)  $22H + 61H + 11H + 26H + 15H + 0DH$

(2)  $35H + 21H + 42H + 54H + 0D8H + 32H$

5. 上机编写调试程序将累加器 A 中的 8 位二进制数转换成 3 位组合 BCD 码。其中,百位数放在内部 RAM 的 31H 单元中,十位和个位放在 32H 单元。

6. 上机编写调试程序,根据 40H 单元存放的变量  $X$ (无符号数)的值,求得函数  $Y$  的

值并存入 31H 单元。

$$\text{其中} \quad Y = \begin{cases} X-1 & X < 6 \\ 0 & 6 \leq X \leq 10 \\ X+1 & X > 10 \end{cases}$$

7. 连接好单片机开发系统,进行在线仿真调试程序,观察 RAM、寄存器等的变化。

8. 连接单片机的晶振电路和复位电路及供电电源,利用示波器测试 XTAL1、XTAL2 管脚波形;改变晶振数值再重新测试,观察区别。改变复位电阻和电容数值,测试 RST 脚的波形,观察电阻、电容变化时复位信号的变化情况。保留调试好的电路,以备后续章节实践训练使用。

9. 熟练掌握编程器操作。编写 P2.5 管脚输出方波程序,首先模拟调试通过,然后利用编程器下载到单片机的 Flash 中,并将单片机放于第 7 题的硬件电路中,利用示波器测试波形。

## 第6章 AT89S52 单片机并行 I/O 接口

单片机需要与外围设备进行信息交换,信息交换通过 I/O 接口实现。随着超大规模集成电路技术的发展,越来越多的功能部件、器件集成到单片机的芯片内,使之功能不断增强,单片机内部集成了并行 I/O 接口电路,用于与外围设备交换信息。本章首先介绍 I/O 接口的基本概念、I/O 接口的控制方式,然后介绍 AT89S52 单片机并行 I/O 接口及其应用。

### 6.1 I/O 接口概述

I/O 接口是 CPU 和外围设备之间交换信息的连接部件,是 CPU 与外设之间进行数据传送的桥梁和纽带。外围设备种类繁多,有机械式、机电式或电子式等,有输入设备、输出设备,外围设备的工作速度通常比 CPU 的速度低很多,且不同外围设备的工作速度、信息类型、传送方式差别很大,因此导致 CPU 与外设之间的信息传送十分复杂,所以 CPU 和外设之间必须有接口电路,通过接口电路协调单片机与外设之间的数据传送。

#### 6.1.1 I/O 接口的功能

在单片机应用中,输入设备通过 I/O 接口电路把程序、数据或现场采集到的各种信息输入单片机,单片机的处理结果和控制信息要通过 I/O 接口电路传送到输出装置,以便显示、打印或实现各种控制。一般来说 I/O 接口电路的主要功能如下。

##### 1. 地址译码

由译码器对地址进行译码,选择外围设备,以便 CPU 对被寻址的外设进行读/写操作。

##### 2. 数据缓冲和锁存

CPU 通过总线与多个外设打交道。但是,各输入设备的数据线不能都直接与 CPU 的数据总线相连,必须经输入缓冲器接到数据总线上;否则,会出现几个输入设备同时占用数据总线,发生“总线冲突”,以致 CPU 不能正常工作。缓冲电路便于实现在同一时刻 CPU 只与一个外设交换信息,即只有被选中的外设与 CPU 交换信息。

单片机传送信息的速度,一般远远高于外设的工作速度。当单片机向外设输出信息时,外设还来不及立即将信息处理完毕。例如,点阵式打印机打印一个字符约需 10 ms,而单片机输出一个字符只需 10  $\mu$ s 左右。因此,在输出接口电路中应设置数据锁存器,以便及时地把 CPU 输出的数据锁存起来,然后再由外设进行处理。

##### 3. 信息转换

外设送往单片机的信息应该转换成单片机所能接收的数字量,而单片机输出的信息应该转换成外设所要求的信号。因此,I/O 接口电路应能实现信息的转换。例如,串行、并行数据的互相转换,电压、电流的转换,电平的转换,模/数的转换,数/模的转换等。

#### 4. 通信联络

为了协调 CPU 与外设之间的信息交换, CPU 需要通过 I/O 接口电路以一定的方式与外设进行通信联络, 以保证不丢失信息。

为了进行通信联络, I/O 接口电路传送的信息除了数据之外, 还要提供外设状态信息, 以及 CPU 对外设的启停控制信号等。

### 6.1.2 接口与端口

I/O 接口的功能主要通过电路实现, 因此也称之为接口电路。在接口电路中应该包含数据寄存器以保存输入输出数据, 状态寄存器以保存外设的状态信息, 命令寄存器以保存来自 CPU 的有关数据传送的控制命令。由于在数据的传送中, CPU 需要对这些寄存器的状态口和保存命令的命令口寻址等, 通常把接口电路中这些已编址并能进行读或写操作的寄存器称为端口(port), 或简称口。因此, 一个接口电路就对应着多个端口地址, 对它们像存储单元一样进行编址。

一个典型的 I/O 接口如图 6.1.1 所示, 其中有数据端口、状态端口和控制端口, 如 8155。CPU 通过这些端口与外部设备之间进行信息的传送。通常将信息按各自的作用分成以下 3 种。

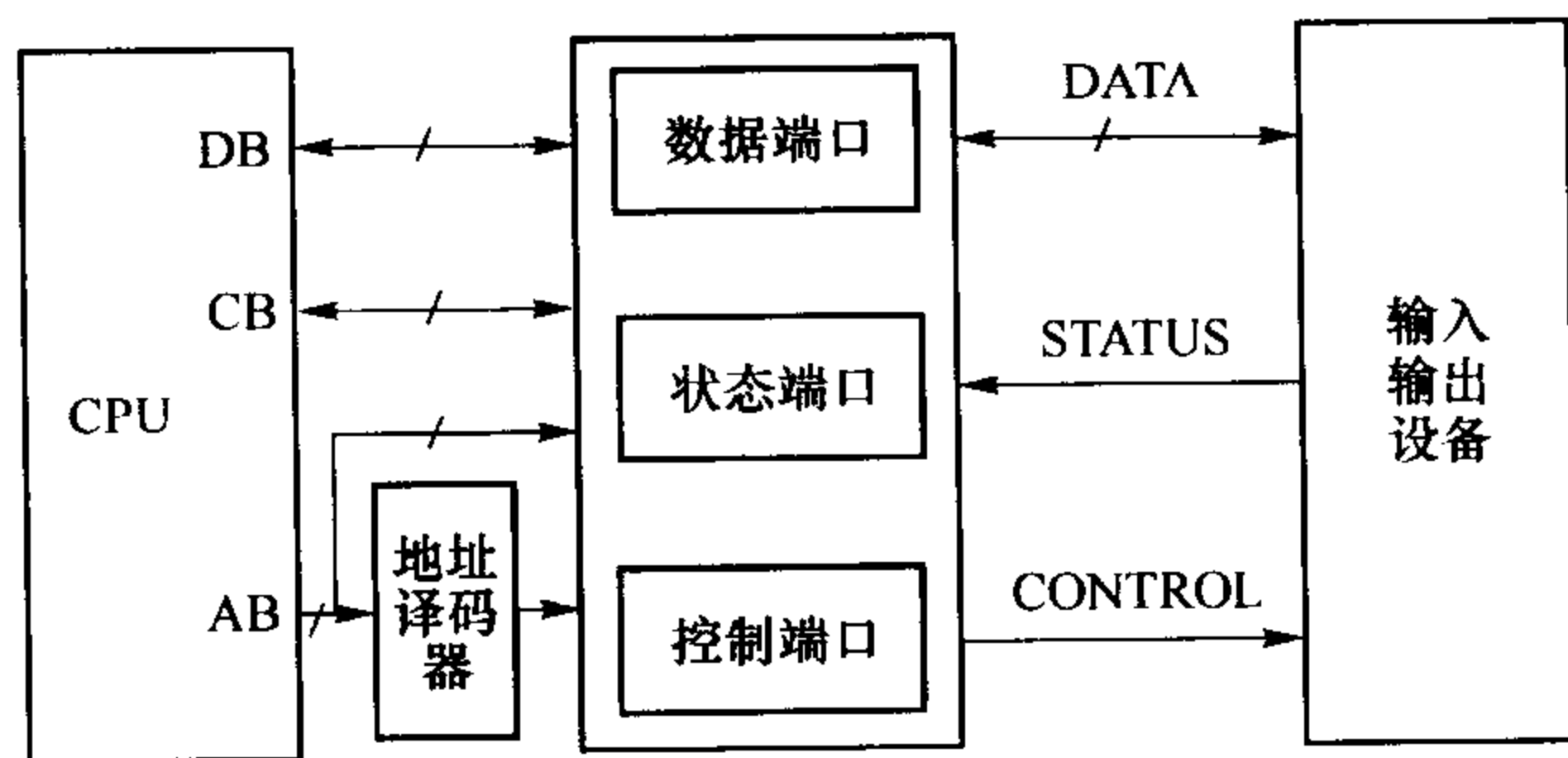


图 6.1.1 I/O 接口的基本构成

#### 1. 数据信息

数据信息是最基本的一种信息, 按其数据的表示形式又可以分为数字量、模拟量。

##### (1) 数字量

用 8 位二进制数或 ASCII 码表示的数据或字符。

开关量是表示两个状态的量, 只用一位二进制数表示。

脉冲量是一个一个传送的脉冲列。计数脉冲、定时脉冲等在单片机控制系统中也很常见。

##### (2) 模拟量

当单片机用于控制、检测或数据采集时, 大量的现场信息是连续变化的物理量(如温度、压力、流量、位移等), 这些物理量经过传感器变换成电量, 并经放大得到模拟电压或电流, 这些模拟量必须再经过 A/D 转换后, 把它们变成数字量才能输入单片机。单片机的输出也必须先经过 D/A 转换, 把数字量变成模拟量后再控制执行机构。



## 2. 状态信息

状态信息反映了外部设备当前所处的工作状态,作为单片机与外设交换信息的联络信号。例如,输入设备的“准备就绪”、输出设备的“忙”信号等。CPU 根据外设的状态,决定是否输入或输出数据。

## 3. 控制信息

控制信息是在传送过程中 CPU 发送给外设的命令,用于控制外设的工作。例如,控制设备的启动或停止等。

数据信息、状态信息和控制信息是不同性质的 3 类信息,应通过不同的端口分别传送。也就是说,CPU 送往外设的数据或者外设送往 CPU 的数据放在数据端口的数据寄存器中,从外设送往 CPU 的状态信息放在状态端口的状态寄存器中,而 CPU 送往外设的控制信息要送到控制端口的控制寄存器中。

### 6.1.3 I/O 接口编址技术

端口是接口电路中能被 CPU 直接访问的寄存器,每个端口通常分配一个端口地址,CPU 通过对端口地址的访问发送命令,读取状态或输入、输出数据。因此,一个接口可以有几个端口,如上所述有数据端口、状态端口和控制端口,CPU 与外设之间就是通过 I/O 接口来进行数据的交换。要实现对这些端口的访问,就应该对端口编址。I/O 接口有以下两种编址方式。

#### 1. I/O 接口和存储器统一编址方式

在这种编址方式中,I/O 接口和存储器共用一个地址空间,端口与存储器统一编址,即把每个端口当成一个存储单元,给它分配存储空间的一个地址。因此,CPU 用访问存储器的指令对 I/O 接口进行读/写操作(MOVX 指令)。

这种方式的主要优点是:CPU 无须专用的 I/O 指令和接口信号,能以丰富的访问存储器指令来访问 I/O 接口,处理能力强。但是端口占用了存储器的地址,指令的执行时间较长。AT89S52 单片机即采用这种方式。

#### 2. I/O 接口独立编址方式

这种方式是端口地址与存储器地址分开编址。对 I/O 接口独立编址,需要专门的 I/O 指令和接口信号访问 I/O 接口。这种方式的主要优点是:处理速度较快,端口地址不占用存储空间,各自都有完整的地址空间。

### 6.1.4 I/O 数据传送的控制方式

单片机与外设之间的数据传送方式可归纳为 3 种:程序传送、中断传送和直接存储器存取传送。

#### 1. 程序传送

程序传送,是指 CPU 与外设之间的数据在程序控制下进行传送的方式,它又分为无条件传送和条件传送两种。

##### (1) 无条件传送

这种传送方式的特点是,数据能否进行传送只取决于程序的执行,而与外设的条件



(即状态)无关。也就是说,在需要进行传送时,认为外设已处于“准备就绪”状态,程序执行 I/O 指令,CPU 就立即与外设进行数据传送。

无条件传送方式,一般用于对固定的输入/输出装置传送单个信息。例如,读开关数据、驱动继电器、驱动七段数码管等。

这种传送方式的优点是接口电路和程序设计都非常简单。其传送的工作原理如图 6.1.2 所示。图中,由于来自输入设备的数据保持时间相对于 CPU 的接收速度要长得多,所以输入数据不需要用锁存器来锁存,而直接使用三态缓冲器与数据总线相连。

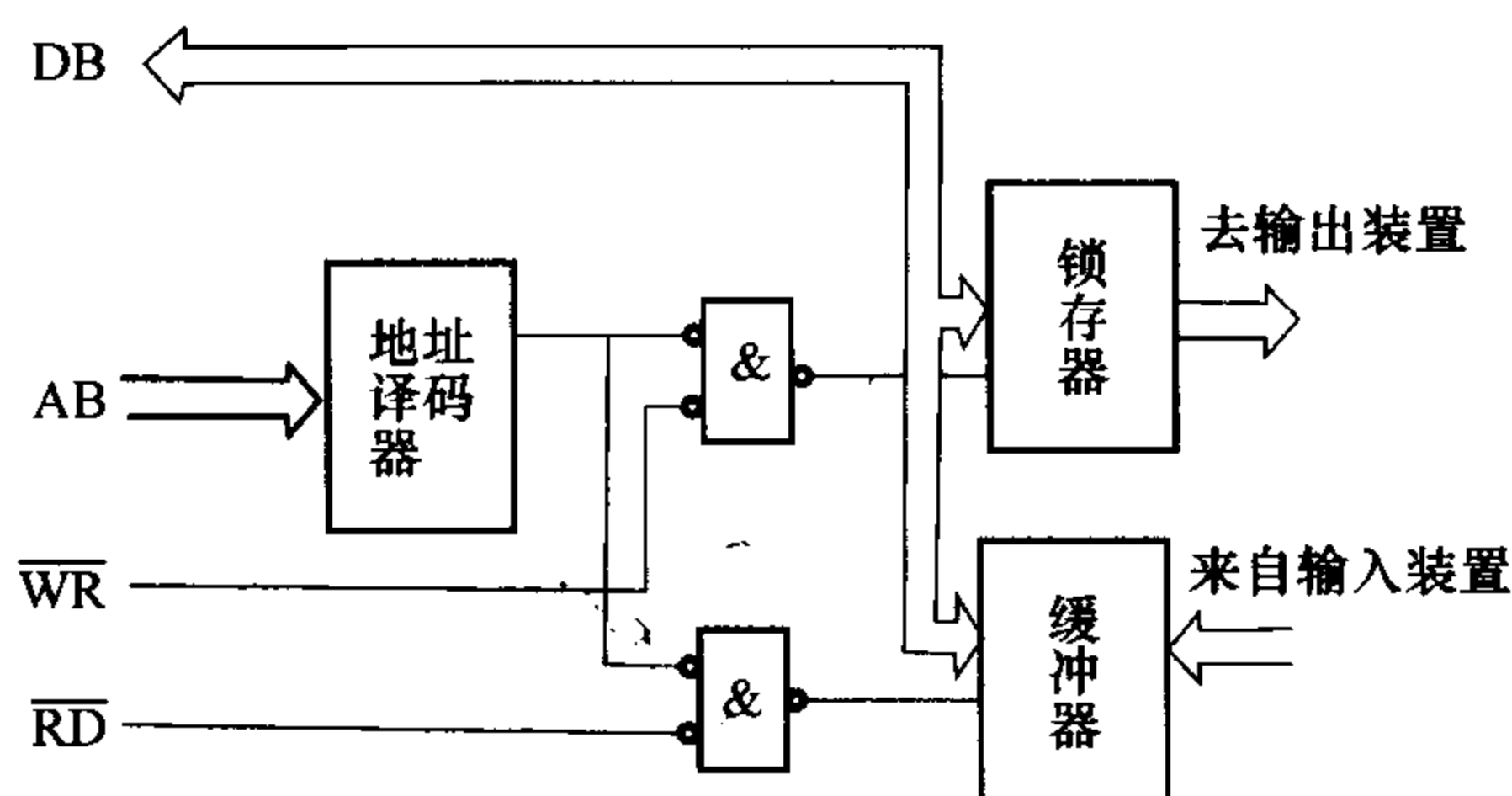


图 6.1.2 无条件传送方式的工作原理

对输出设备,一般都需要用锁存器来锁存 CPU 送出的数据,使其保持一定的时间,以便与外设的动作相适应。

## (2) 条件传送

条件传送又称查询传送(Polling)。传送前,CPU 读取外设的状态,并加以测试判断,如果外设“准备就绪”,则 CPU 就向外设传送数据;如果外设未“准备就绪”,则不进行数据传送,CPU 转而继续查询外设的状态。

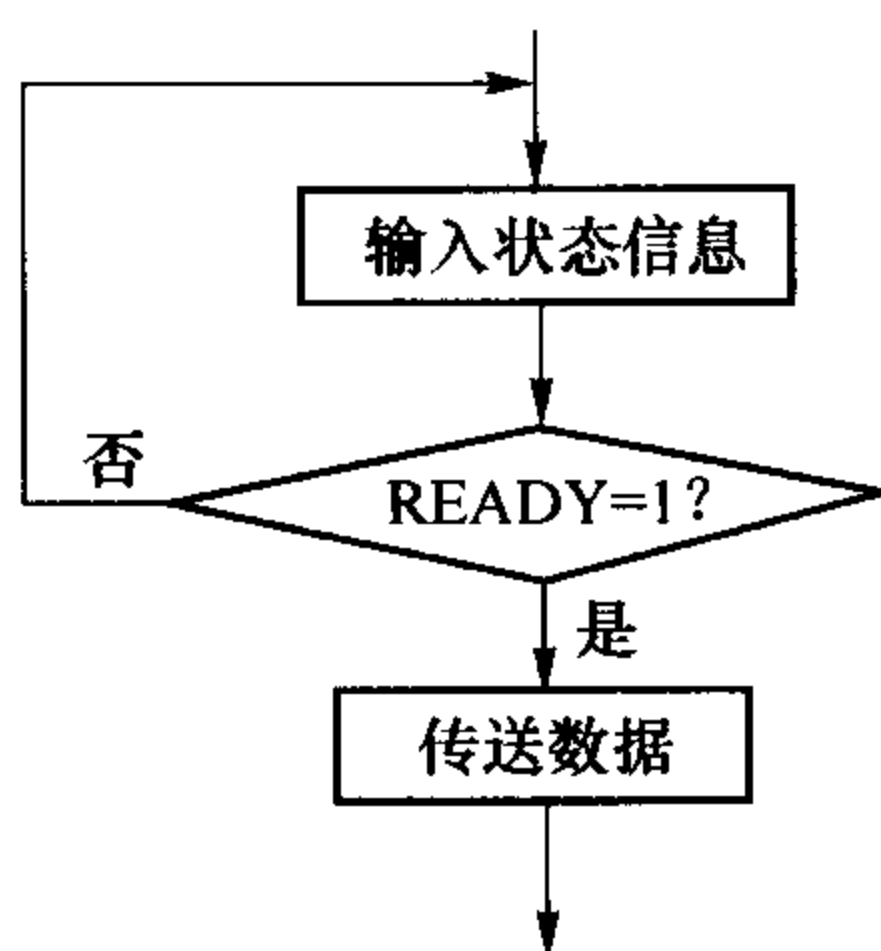


图 6.1.3 查询方式传送的流程图

所谓“准备就绪”,对于输入设备而言,即输入数据寄存器已满,准备好新数据供 CPU 读取;对于输出设备而言,即输出数据寄存器已空,可以接收 CPU 送来的新数据。

查询方式传送数据的流程图如图 6.1.3 所示。由图可见,以查询方式传送数据的步骤如下:

- ① CPU 读取外设状态信息;
- ② CPU 检测外设状态是否满足“准备就绪”条件,如果不满足,则回到①继续读取状态信息;
- ③ 如果外设状态为“准备就绪”,则传送数据。

以查询方式接收输入信息的接口电路如图 6.1.4 所示。输入设备准备好后发一个选通信号。它一方面把输入设备准备传送的数据存入锁存器;另一方面使 D 触发器置 1,作为准备好的一个状态信号。数据信息和状态信息从不同的端口经过数据总线送到 CPU。当 CPU 要从外设输入数据时,先读入状态信息,检测数据是否准备好,若准备就绪,就执

行传送指令读取数据,同时使状态信息(READY)清零,以便传送下一个数据。

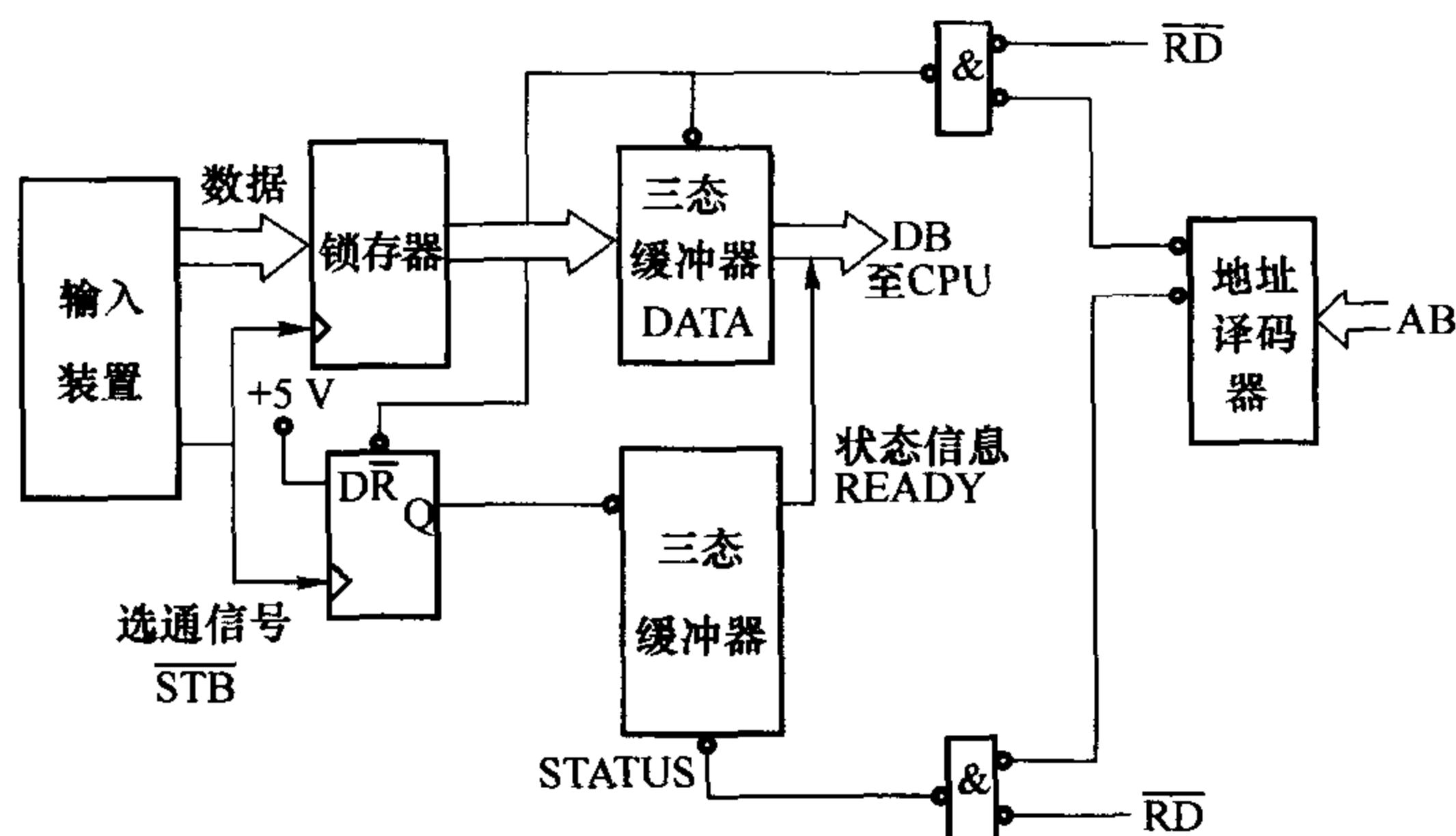


图 6.1.4 查询式输入接口电路

以查询方式输出的接口电路如图 6.1.5 所示。当 CPU 要向外设输出数据时,CPU 应先读取外设的状态信息,如果外设不忙 ( $BUSY = 0$ ),则表明可以向外设输出数据,CPU 执行传送指令,把数据送出;否则,CPU 就继续查询等待。

当输出设备接收 CPU 输出的数据以后,发出一个应答信号,使 D 触发器清 0,表示外设已空;当 CPU 读取这个状态信息并判断外设已空时,便执行输出操作,由地址信号和写信号产生选通信号把 CPU 输出的数据打入锁存器,同时使 D 触发器置 1。D 触发器的输出信号 ( $BUSY$ ) 一方面为外设提供了一个联络信号,通知外设输出数据已准备好,可供提取;另一方面告诉 CPU,当前外设处于“忙”状态,从而阻止 CPU 输出新的数据。

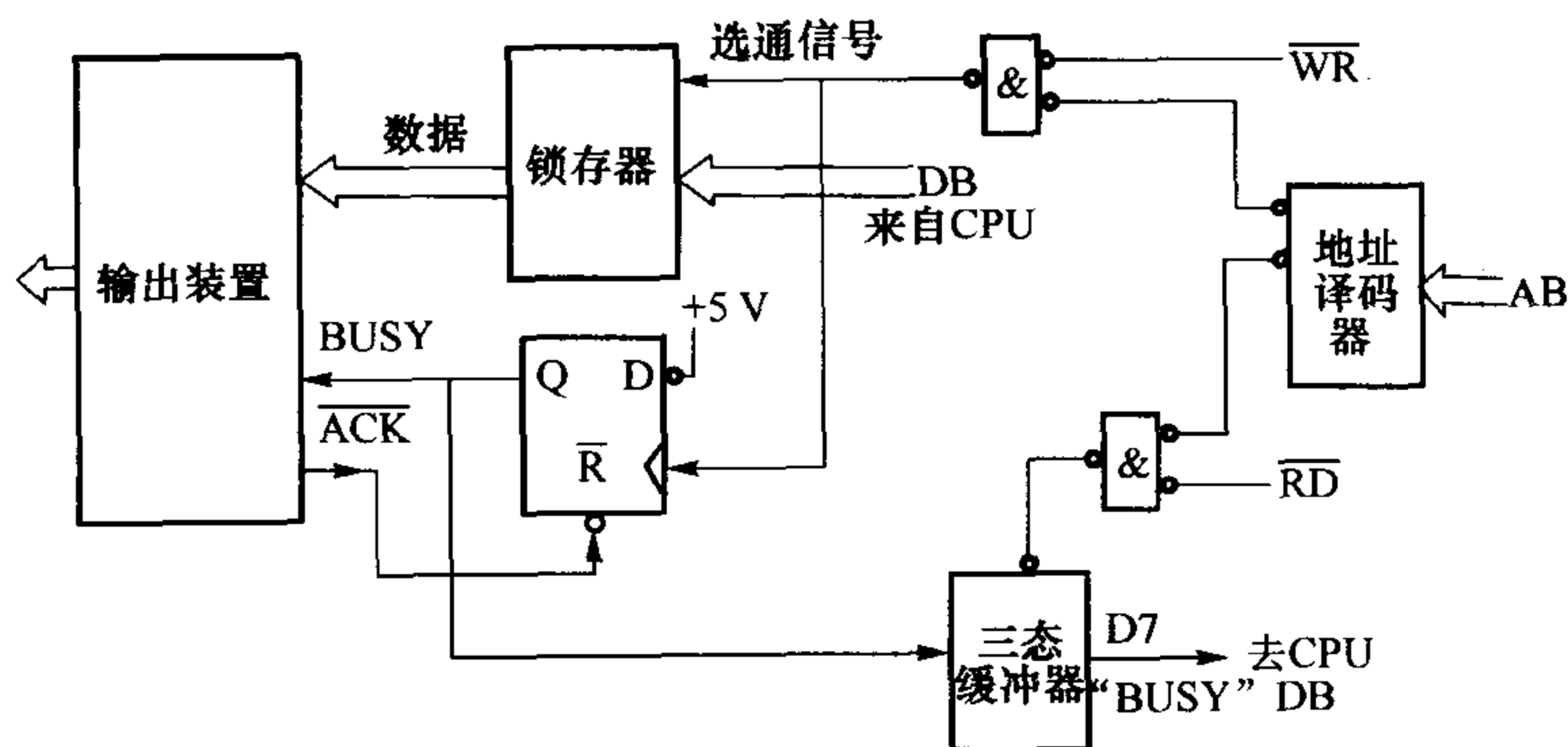


图 6.1.5 查询式输出接口电路

## 2. 中断传送

为了提高 CPU 的运行效率和使系统具有实时性能,可采用中断传送方式。在这种传送方式下,每个外设都具有请求 CPU 服务的主动权,可以随机地向 CPU 提出中断申请,而 CPU 又能在每一条指令执行的结尾阶段检查外设是否有中断请求。因此,如果没有中断申请发生,CPU 执行与外设无关的操作。一旦外设需要服务,就主动向 CPU 发出中断请求,CPU 便可暂时中止当前执行的程序,而转去为外设服务,进行一次数据传送;

服务完以后,CPU 返回断点继续执行原来的程序。这样,CPU 不必浪费大量的时间去轮流查询各个外设的状态,从而可提高 CPU 的工作效率,同时也使系统具有很强的实时性能。

### 3. 直接存储器存取传送

单片机同外设交换信息,实际上是单片机的内部 RAM 同外设交换信息,因为信息保存在 RAM 中。单片机输出时,CPU 从 RAM 中读取数据,然后通过输出接口将信息传送给外设;输入时,由外设经过输入接口把信息经 CPU 装入 RAM。

中断传送方式虽然大大提高了 CPU 的利用率,但它仍是由 CPU 通过程序传送,每次传送前要保护断点、保护现场,传送后又要恢复现场和断点,因而需要执行多条指令。通常用中断方式传送一个字节数据需要几十微秒,甚至更多的时间,这对于一个高速输入/输出设备及需要成组交换数据的场合(例如磁盘与 RAM 间的信息交换),就显得速度太慢了。

直接存储器存取(DMA, Direct Memory Access)是一种由专门的硬件 DMA 控制器(DMAC)来控制的传送方式。在 DMAC 的控制下,外设接口可直接与 RAM 进行高速的数据传送,而不必经过 CPU,于是进行传送时就不必做保护现场、恢复现场之类的额外操作。DMA 方式主要是通过硬件实现,因而传送速率很高,数据传送的速度基本上取决于外设和存储器的速度。DMA 方式特别适用于大批量数据的高速传送。

## 6.2 AT89S52 并行 I/O 接口的内部结构

AT89S52 内部集成 4 个并行 I/O 接口,称为 P0, P1, P2 和 P3,共占用 32 只引脚,每一条 I/O 线都能独立地作为输入或输出,但 4 个通道的功能不完全相同。当无须外部扩展时,P0, P1, P2 均可作为典型的并行 I/O 接口使用,P3 口作为 I/O 接口或第二功能口用。当需要外部扩展时,P0 口为地址/数据分时复用,P2 口为高 8 位地址线,由 P0 和 P2 口组成 16 位地址线。

### 6.2.1 I/O 接口的结构特点

#### 1. 锁存器加引脚的典型结构

AT89S52 单片机的 I/O 接口都由内部总线实现操作控制。P0~P3 这 4 个 I/O 接口都可作为普通 I/O 口,因此,要求有输出锁存功能。内部总线又是分时操作,故每个 I/O 接口都有相应的锁存器。然而,I/O 接口又是与外部电路连接的输入/输出通道,必须有相应的引脚,故形成了 I/O 接口的锁存器加引脚的典型结构。

#### 2. I/O 接口的复用结构

(1) I/O 接口的总线复用。AT89S52 单片机在使用并行扩展总线时,P0 口可作为数据总线口和低 8 位地址总线口,这时,P0 口为真正双向口。P0 口输出并行总线的地址/数据信号,P2 口输出高 8 位地址信号。

(2) I/O 接口的功能复用。AT89S52 单片机的 P3 口、P1 口(低两位)为功能复用的 I/O 接口。接口内部有复用输出功能的控制端,引脚也有复用输入功能的控制端。

### 3. 准双向口结构

P0, P1, P2, P3 口作为普通 I/O 口使用时, 都是准双向口结构。准双向口的输入操作和输出操作的本质不同, 输入操作是读引脚状态, 输出操作是对口锁存器的写操作。输出操作时即当由内部总线给口锁存器置 0 或 1 时, 锁存器中的 0, 1 状态立即反映到引脚上。但是在输入操作(读引脚)时, 如果口锁存器状态为 0, 引脚被钳位在 0 状态, 导致无法读出引脚的高电平输入。这就要求在输入操作前先置位相应锁存器, 因此称为准双向口(参见 6.2.2 小节)。

## 6.2.2 AT89S52 的并行 I/O 接口

AT89S52 单片机共有 4 个 8 位的并行双向口, 共有 32 条 I/O 口线。由于结构上的一些差异, 故各口的性质和功能也就有了差异。它们之间的异同如表 6.2.1 所示。

表 6.2.1 AT89S52 并行 I/O 接口

I/O 接口	P0 口	P1 口	P2 口	P3 口
性质	真正双向口	准双向口	准双向口	准双向口
功能	I/O 接口, 第二功能	I/O 接口, 第二功能	I/O 接口, 第二功能	I/O 接口, 第二功能
SFR 字节地址	80H	90H	A0H	B0H
位地址范围	80H~87H	90H~97H	A0H~A7H	B0H~B7H
驱动能力	8 个 TTL 负载	4 个 TTL 负载	4 个 TTL 负载	4 个 TTL 负载
第二功能	程序存储器、片外数据存储器低 8 位地址及 8 位数据	CTC2, T2, T2EX	程序存储器、片外数据存储器高 8 位地址	串行口: RXD, TXD 中断: $\overline{\text{INT0}}$ , $\overline{\text{INT1}}$ 计数器: T0, T1 读写信号: $\overline{\text{WR}}$ , $\overline{\text{RD}}$

### 1. P0 口

P0 口是一个多功能的 8 位口, 可以字节访问也可位访问, 其字节访问地址为 80H, 位访问地址为 80H~87H。P0 口包含输出锁存器, 输入缓冲器 BUF1(读引脚), BUF2(读锁存器)以及由 FET、Q1 和 Q0 组成的上拉和下拉电路。

#### (1) 位结构和工作原理

P0 口位结构原理如图 6.2.1 所示, 工作过程分析如下:

① P0 口中多路开关的输入有两个: 地址/数据输出, 输出锁存器的输出  $\overline{Q}$ 。多路开关的输出用于控制输出 FET Q0 的导通和截止。多路开关的切换由内部控制信号控制。

② P0 口的输出上拉电路 Q1 的导通和截止受内部控制信号和地址/数据信号共同(相“与”)控制。

③ 当控制信号置 1 时, 多路开关接通地址/数据输出端。

地址/数据输出线置 1 时, 控制上拉电路的“与”门输出为 1, 上拉 FET 导通, 同时地址/数据输出通过反相器输出 0, 下拉 FET 截止, 这样 A 点电位上拉, 地址/数据输出线为 1。

地址/数据输出线置 0 时, “与”门输出为 0, 上拉 FET 截止, 同时地址/数据输出通过反相器输出 1, 控制下拉 FET 导通, 这样 A 点电位下拉, 地址/数据输出线为 0。

通过上述分析可以看出,此时的输出状态随地址/数据线而变。因此,P0口可以作为地址/数据复用总线使用。这时上下两个FET处于反相,构成了推拉式输出电路,其负载能力增加。此时的P0口相当于一个双向口。

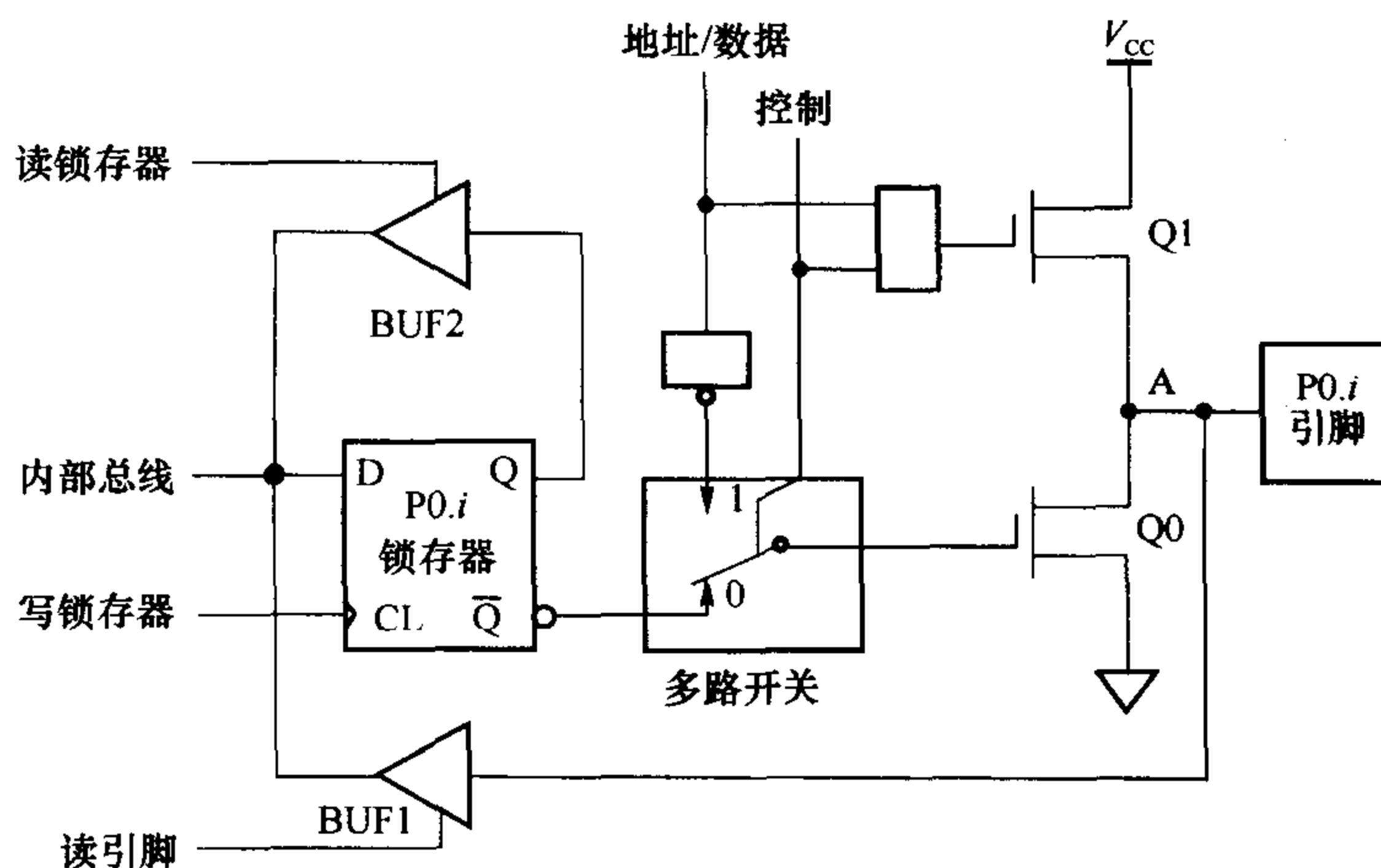


图 6.2.1 P0 口位结构原理图

④ 当控制信号置 0 时,多路开关接通输出锁存器的  $\bar{Q}$  端。这时明显地可以看出两点:

- 由于内部控制信号为 0,与门关闭,上拉 FET 截止,形成 P0 口的输出电路为漏极开路输出,故需外接上拉电阻才能正常工作;
- 输出锁存器的  $\bar{Q}$  端引至下拉 FET 栅极,因此 P0 口的输出状态由下拉电路决定。

P0 口作输出口用时,若 P0.i 输出 1,内部总线上该位为 1,即  $D=1$ ,输出锁存器的  $\bar{Q}$  端为 0,下拉 FET 截止,这时 P0.i 为漏极开路输出;若 P0.i 输出 0,则  $D=0$ ,输出锁存器的  $\bar{Q}$  端为 1,下拉 FET 导通,P0.i 输出低电平。

P0 口作输入口用时,为了能正确读入 P0.i 数据,必须先使 P0.i 置 1,此操作设置 P0.i 为输入线。这样,下拉 FET 也截止( $D=1, \bar{Q}=0$ ),P0.i 处于悬浮状态。A 点的电平由外设的电平而定,通过输入缓冲器读入 CPU。这时 P0 口相当于一个高阻抗的输入口。

## (2) P0 口的功能

### ① 作 I/O 接口使用

P0 口相当于一个准双向口:输出锁存、输入缓冲,但输入时需先将其置 1;每条口线可以独立定义为输入或输出,具有双向口的一切特点。

与其他口的区别是,漏极开路输出,与其他电路接口时必须要用上拉电阻,才能有高电平输出;输入时为悬浮状态,为一个高阻抗的输入口。

### ② 作地址/数据复用总线用

地址/数据输出时,控制信号为 1,输出状态随地址/数据线变化。数据输入时,控制信号为 0,上拉 FET 截止,CPU 自动向 P0 口写 0FFH, $\bar{Q}=0$ ,下拉 FET 截止,保证数据高阻输入。此时 P0 口是一个真正的双向口。P0 口作地址/数据复用之后,就不能再作 I/O 接口使用。

应该注意:当前许多仿真开发系统中,均以 P0 口作地址/数据复用总线使用,因而不能仿真 I/O 接口的功能。

## 2. P1 口

P1 口是一个 8 位口,可以字节访问也可按位访问,其字节访问地址为 90H,位访问地址为 90H~97H。

### (1) 位结构和工作原理

P1 口的位结构如图 6.2.2 所示。

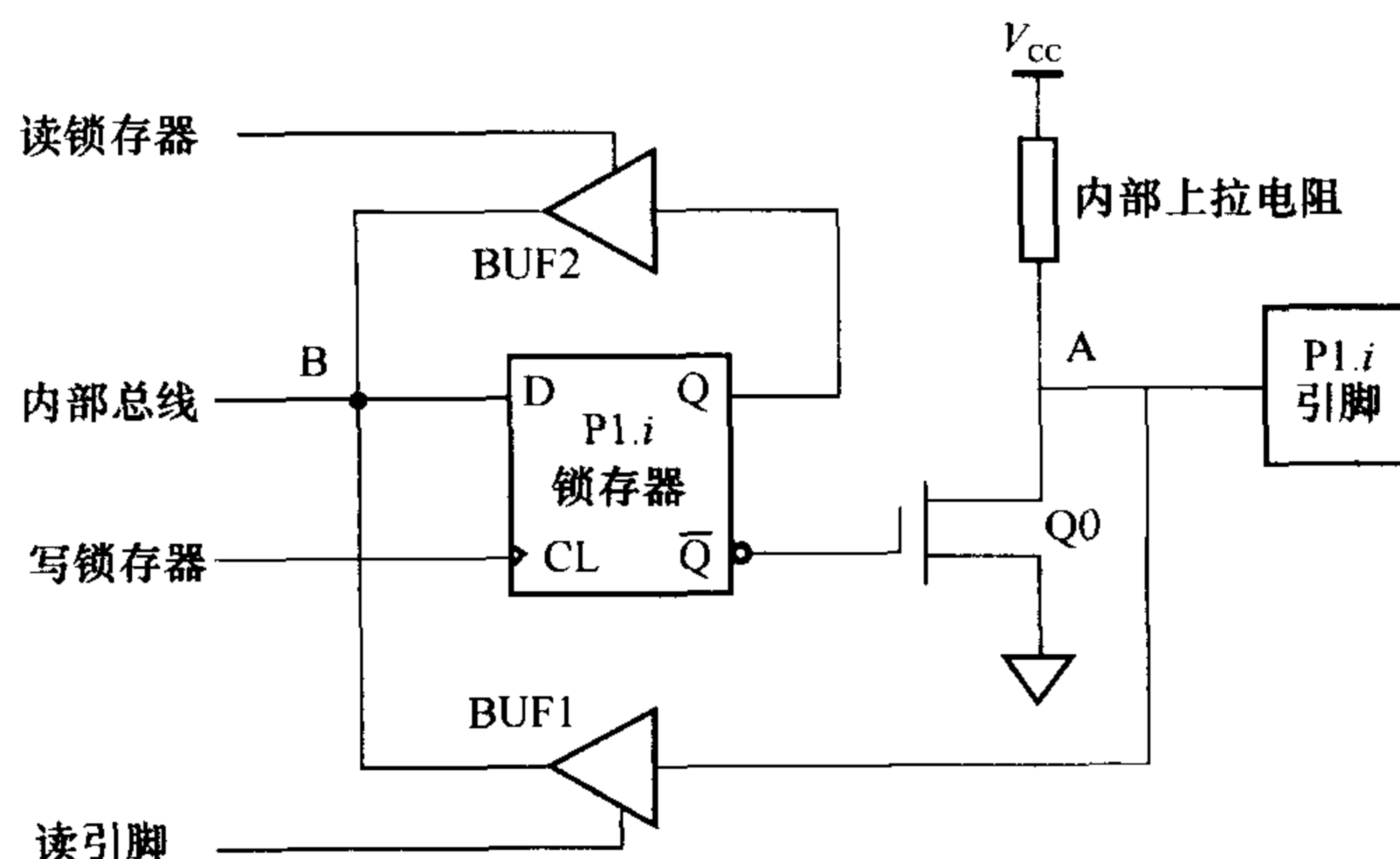


图 6.2.2 P1 口位结构原理图

包含输出锁存器、输入缓冲器 BUF1(读引脚)、BUF2(读锁存器)以及由 FET 晶体管 Q0 与上拉电阻组成的输出/输入驱动器。P1 口的工作过程分析如下。

#### ① P1 口作输出口

CPU 输出 0 时, $D=0, Q=0, \bar{Q}=1$ , 晶体管 Q0 导通, A 点被下拉为低电平, 即输出 0; CPU 输出 1 时, $D=1, Q=1, \bar{Q}=0$ , 晶体管 Q0 截止, A 点被上拉为高电平, 即输出 1。

#### ② P1 口作输入口

先向 P1.i 位输出高电平( $D=1, \bar{Q}=0$ ), Q0 截止, 使 A 点提升为高电平, 此操作设置 P1.i 为输入线。若外设输入为 1 时 A 点为高电平, 由 BUF1 读入总线后 B 点也为高电平; 若外设输入为 0 时 A 点为低电平, 由 BUF1 读入总线后 B 点也为低电平。

### (2) P1 口的功能

- 输出锁存器, 输出时没有条件;
- 输入缓冲, 输入时有条件, 即需要先将该口设为输入状态, 先输出 1;
- 工作过程中无高阻悬浮状态, 也就是该口只有输入和输出两种状态。

具有这种特性的口不属于“真正”的双向口, 而被称为“准”双向口。

这里需要注意的是, 若在输入操作之前不将 A 点设置为高电平(即先向该口线输出 1), 如果 A 点电平为低电平时, 则外设输入的任何信号均被拉为低电平, 亦即此时外设的任何信号都输不进来。更为严重的是, A 点为低电平, 而外设为高电平时, 外设的高电平通过 Q0 强迫下拉为低电平, 将可能有很大的电流流过 Q0 而将它烧坏。

### (3) P1 口的第二功能

在 AT89S52 中, P1.0 和 P1.1 口线是多功能的, 即除作一般双向 I/O 口线之外, 这两条口线还具有下列功能:

- P1.0——定时/计数器 2 的外部事件计数输入端;
- P1.1——定时/计数器 2 的捕捉/重装控制端。

这时, 该两位的结构与 P3 口的位结构相同。

## 3. P2 口

P2 口是一个多功能的 8 位口, 可以字节访问也可位访问, 其字节访问地址为 A0H, 位访问地址为 A0H~A7H。

### (1) P2 口位结构和工作原理

P2 口位结构原理如图 6.2.3 所示。

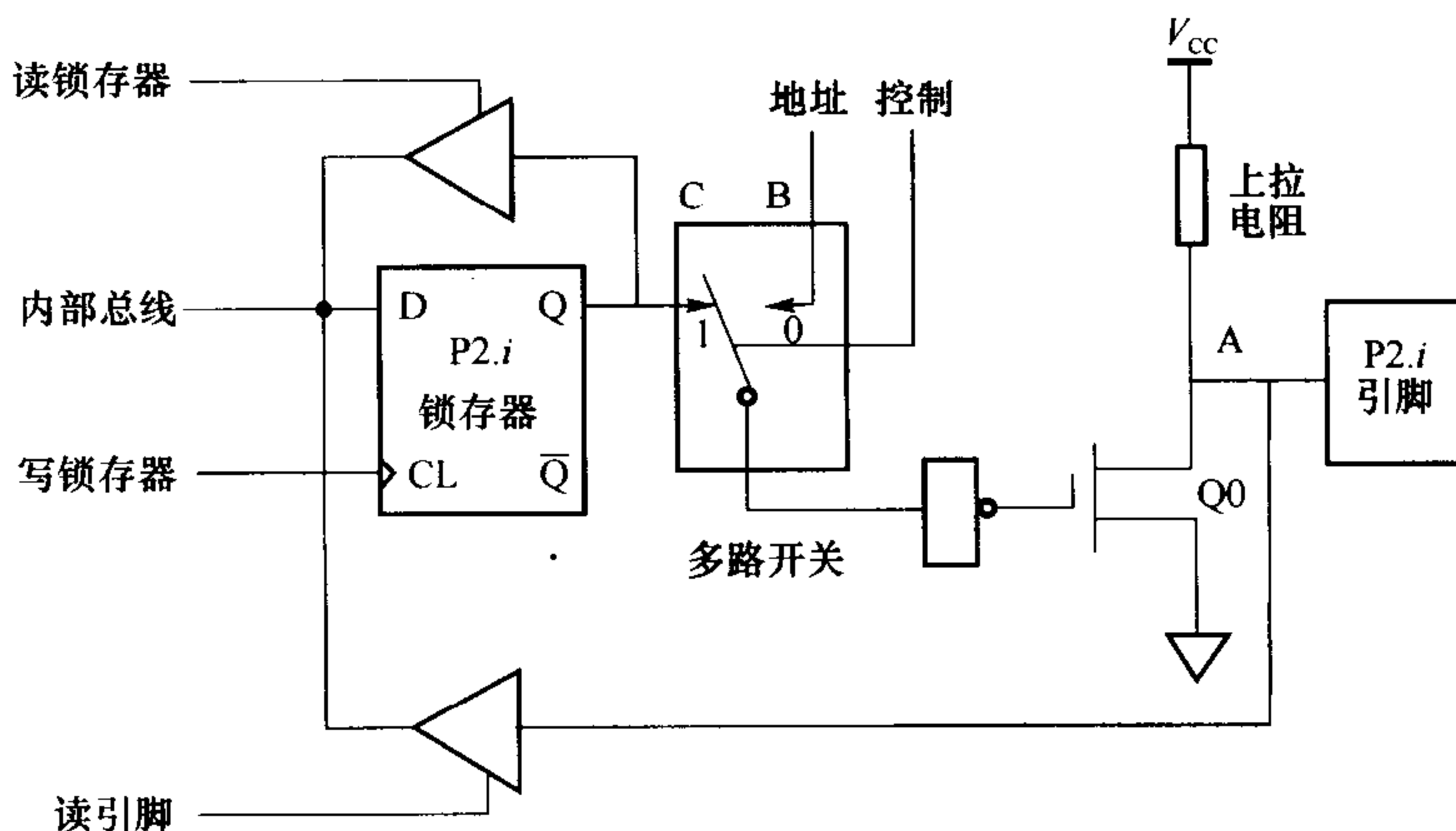


图 6.2.3 P2 口位结构原理图

P2 口与 P1 口位结构基本相同, P2 口的位结构中增加了一个多路开关和一个反相器。

多路开关的输入有两个: 一个是口输出锁存器的输出端 Q, 一个是地址寄存器(PC 或 DPTR)的高位输出端。多路开关的输出经反相器反相后控制输出 FET Q0。多路开关的切换由内部控制信号控制。

### (2) P2 口的功能

从上述工作过程的分析中可以看出 P2 口是一个双功能口:

- ① 作 I/O 口使用时, P2 口为准双向口, 功能与 P1 口一样;
- ② 作地址输出时, P2 口可以输出高 8 位地址。

### (3) P2 口使用中要注意的问题

① 由于 P2 口的输出锁存功能, 在取指周期内或外部数据存储器读、写选通期间, 输出的高 8 位地址是锁存的, 故无须外加地址锁存器。

② 系统中如果外接程序存储器, 由于访问片外程序存储器的连续不断的取指操作, P2 口需要不断送出高位地址, 这时 P2 口的全部口线均不宜再作 I/O 口使用。

③ 在无外接程序存储器而有片外数据存储器的系统中, P2 口的使用可分为两种情况。

- 若片外数据存储器的容量小于 256 B: 可使用 MOVX A, @Ri 及 MOVX @Ri, A 类指令访问片外数据存储器, 这时 P2 口不输出地址, P2 口仍可作为 I/O 接口使用。
- 若片外数据存储器的容量大于 256 B: 这时可使用 MOVX A, @DPTR 及 MOVX @DPTR, A 类指令访问片外数据存储器, P2 口需输出高 8 位地址。在片外数据存储器读、写选通期间, P2 口引脚上锁存高 8 位地址信息, 但是在选通结束后, P2 口内原来锁存的内容又重新出现在引脚上。

使用 MOVX A, @Ri 及 MOVX @Ri, A 类访问指令时, 高位地址通过程序设定, 只利用 P1, P3 甚至 P2 口中的某几条口线送高位地址, 从而保留 P2 口的全部或部分口线作 I/O 接口用。

#### 4. P3 口

P3 口是一个多功能 8 位 I/O 接口, 可以字节访问也可位访问, 其字节访问地址为 B0H, 位访问地址为 B0H~B7H。

##### (1) 位结构与工作原理

P3 口的位结构原理如图 6.2.4 所示。

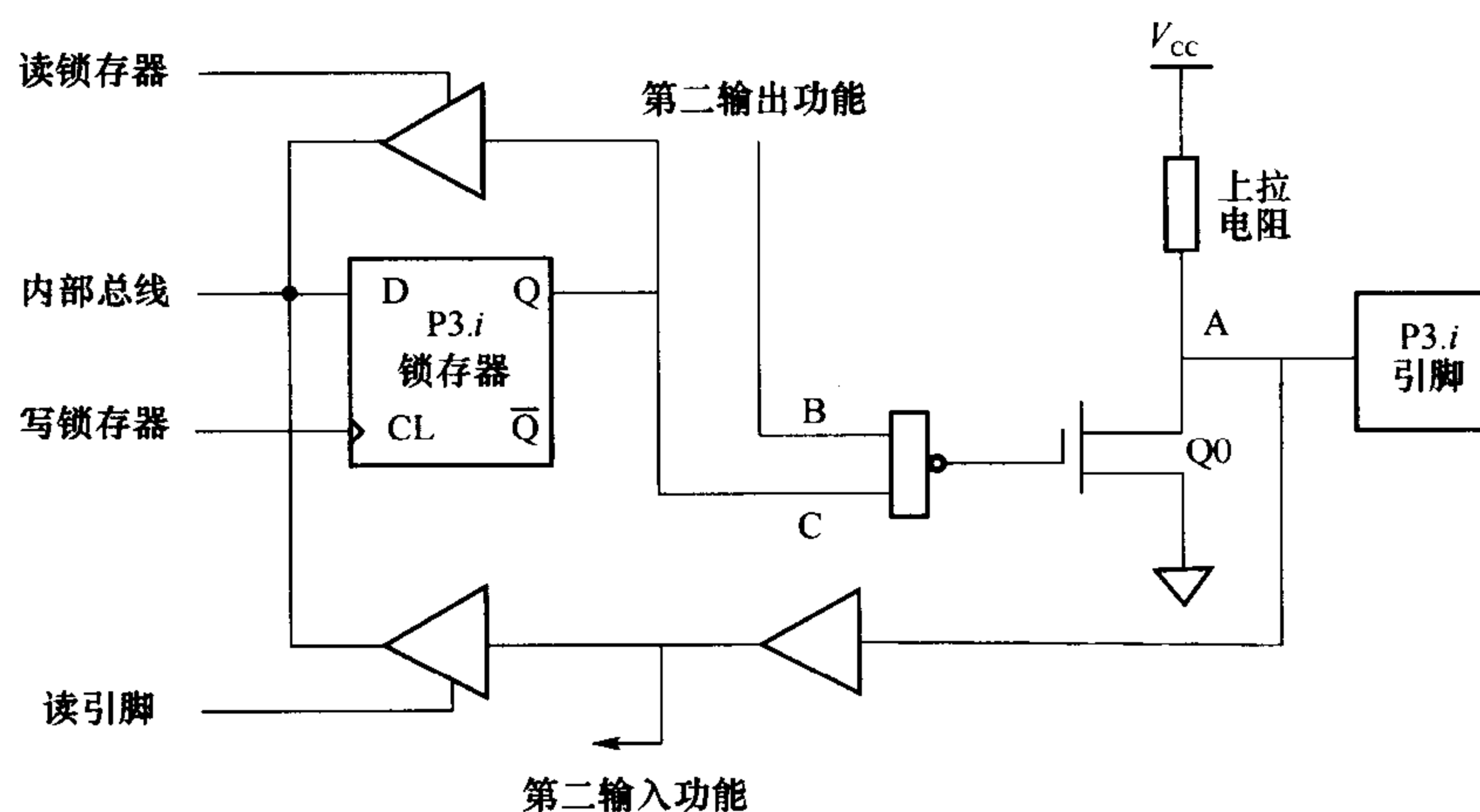


图 6.2.4 P3 口位结构原理图

从 P3 口的位结构图中可以看出, 它与 P1 口位结构基本相同, 相比于 P1 口增加了一个与非门和一个输入缓冲器。与非门有两个输入端: 一个为口输出锁存器的 Q 端, 另一个为第二功能的控制输出。与非门的输出端控制输出 FET 管 Q0。有两个输入缓冲器, 第二输入功能取自第一个缓冲器的输出端; I/O 接口的通用输入信号取自第二个缓冲器的输出端。其工作过程如下。

当第二输出功能 B 点置 1 时, 与非门的输出由输出锁存器决定, 所以输出锁存器的输出可以顺利通到引脚 P3.i。其工作状态与 P1 口相类似。这时 P3 口的工作状态为 I/O 接口, 该口具有准双向口性质。

当输出锁存器的输出置 1 时, 与非门的输出由第二输出功能决定, 所以第二输出功能可以顺利通到引脚 P3.i。若第二输出功能为 0 时, 因与非门的 C 点已置 1, 现 B 点为 0,



故与非门的输出为 1,使 Q0 导通,从而使 A 点也为 0。若第二输出功能为 1 时,与非门的输出为 0,Q0 截止,从而使 A 点也为高电平。这时 P3 口的工作状态处于第二输出功能状态。

### (2) P3 口的功能

与 P1 口不同,P3 口除作 I/O 接口,还是一个多功能口。

① 可作 I/O 接口使用,为准双向口。

② 可以作为第二功能的输入、输出口使用。

第二输入功能:

P3.0——RXD,串行输入口。

P3.2—— $\overline{\text{INT0}}$ ,外部中断 0 的请求。

P3.3—— $\overline{\text{INT1}}$ ,外部中断 1 的请求。

P3.4——T0,定时/计数器 0 外部计数脉冲输入。

P3.5——T1,定时/计数器 1 外部计数脉冲输入。

第二输出功能:

P3.1——TXD,串行输出口。

P3.6—— $\overline{\text{WR}}$ ,外部数据存储器写选通,输出,低电平有效。

P3.7—— $\overline{\text{RD}}$ ,外部数据存储器读选通,输出,低电平有效。

## 6.3 并行 I/O 接口操作

由上述分析可知,AT89S52 的 P0,P1,P2,P3 口均可进行字节操作和位操作,既可以 8 位一组进行输入、输出操作,也可以逐位分别定义各口线为输入线或输出线。由于 AT89S52 采用统一编址方式,因此没有专门的 I/O 指令,4 个 I/O 接口均属于内部的 SFR。其数据传送、位操作等指令如表 6.3.1 所示。

### 1. 有关 I/O 接口操作的指令

表 6.3.1 有关 I/O 接口操作的指令表

数据传 送指令	MOV Px, #DATA		MOV A, Px	
	MOV Px, A		MOV Rn, Px	
	MOV Px, Rn		MOV @Ri, Px	
	MOV Px, @Ri		MOV direct, Px	
	MOV Px, direct			
位操作 指令	MOV Px.y, C	位传送指令	CPL Px.y	位取反
	CLR Px.y	位清 0 指令	JB Px.y, rel	位为 1 转移
	SETB Px.y	位置 1 指令	JBC Px.y, rel	位为 1 转移并清 0
	CPL Px.y	位取反	JNB Px.y, rel	位为 0 转移
逻辑运算 操作指令	ANL Px, A	逻辑与指令	INC Px	加 1 指令
	ORL Px, A	逻辑或指令	DEC Px	减 1 指令
	XRL Px, A	逻辑异或指令	DJNZ Px, rel	减 1 判零条件转移指令

例如:

```
ORL      P1, #00000010 B
```

可以使 P1.1 位口线输出 1,而使其余各位不变。

```
ANL      P1, #11111101 B
```

可以使 P1.1 位口线输出 0,而使其余各位不变。

## 2. I/O 接口的读-修改-写操作

每个并行 I/O 接口均有两种“读”方式:读锁存器和读引脚。在 AT89S52 单片机的指令系统中,有些指令是读锁存器内容,有些指令则是读引脚内容。读锁存器指令,是读取锁存器内容进行处理,再把处理后的值写入锁存器中,这类指令称为“读-修改-写”操作。当指令的目的操作数单元为某 I/O 接口或某 I/O 接口的某位时,该指令所读的是锁存器内容,而不是读引脚上的内容。现列举部分具有此功能的指令。

ANL(逻辑与指令)	例如:ANL P1,A
ORL(逻辑或指令)	例如:ORL P2,A
XRL(逻辑异或指令)	例如:XRL P3,A
CPL(位取反指令)	例如:CPL P3. 0
INC(增量指令)	例如:INC P2
⋮	

如果某个 I/O 接口被指定为源操作数,则为读引脚的操作指令。例如,执行 MOV A,P1时,P1 口的引脚状态传送到累加器中;而相对应的 MOV P1,A 指令则是将累加器的内容传送到 P1 口锁存器中。

之所以要设置读锁存器功能,是因为在某种特定场合,锁存器内容和引脚上的内容有可能不一致。例如,某 I/O 接口的某引脚与驱动三极管的基极相连,如图 6.3.1 所示。当该位写入 1 而使晶体三极管导通,此时引脚(基极)上的电压发生了变化,从 1 变成了 0,而锁存器中的内容仍为 1,CPU 从两者读取的内容是不一致的,这种变化是符合实际的。这说明,在某种特定场合,输出和输入信号在引脚上会发生变化,产生不一致。为避免这种错误,故设置了读锁存器和读引脚两种方式。

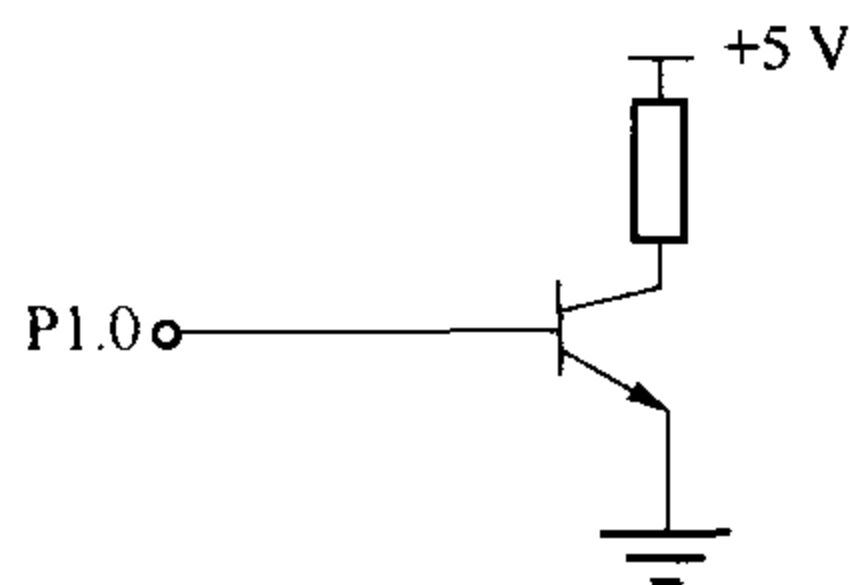


图 6.3.1 P1.0 与三极管的接口电路

## 6.4 I/O 接口应用

### 6.4.1 I/O 接口应用特性

AT89S52 单片机内部有 4 个 8 位双向输入/输出口,它们的内部结构已在前面作过介绍。从特性上看,这 4 个接口还有所差别。

P0 口:除了作为 8 位 I/O 接口外,在扩展外部程序存储器和数据存储器时,P0 口要作为低 8 位地址总线和 8 位数据总线用。即在这种情况下,P0 口不能作 I/O 接口用,而

是先作为地址总线对外传送低 8 位地址,然后作为数据总线对外交换数据。

P1 口:只有 I/O 接口功能,没有其他功能(除 P1.0, P1.1 外)。故在任何情况下, P1 口都可以作 I/O 接口用。

P2 口:在扩展外围设备时,要作为高 8 位地址线用。

P3 口:每个引脚都有不同的第二功能。当某些引脚用作第二功能时,这些口线将不能作为 8 位 I/O 接口。

综上所述, AT89S52 单片机的 I/O 接口具有以下应用特性。

(1) 端口自动识别。无论是 P0, P2 口的总线复用, 还是 P3 口的功能复用, 内部资源会自动选择, 不需要通过指令的状态选择。

(2) 口锁存器的读、改、写功能。许多涉及到 I/O 接口的操作, 实际上只是涉及口锁存器的读出、修改、写入的操作。这些指令都是一些逻辑运算指令、置位、清除指令、条件转移指令以及将 I/O 接口作为目的地址的操作指令。

(3) 准双向口功能。由准双向口的结构可知, 准双向口作输入口时, 应先使锁存器置 1, 然后再读引脚。例如, 要将 P1 端口状态读入到累加器 A 中, 应执行以下两条指令:

```
MOV    P1, #0FFH        ;P1 口置输入方式
MOV    A, P1             ;读 P1 口引脚状态到 A 中
```

(4) P0 口作普通 I/O 接口使用。当不使用并行扩展总线时, P0, P2 口都可用作普通 I/O 接口。但 P0 口为开漏结构, 作 I/O 接口时必须外加上拉电阻。

(5) I/O 口的驱动特性。P0 口每一个 I/O 接口可输出驱动 8 个 LSTTL 输入端, 而 P1~P3 口则可驱动 4 个 LSTTL 输入端。CMOS 单片机的 I/O 接口通常只能提供几毫安的驱动电流, 在全 CMOS 应用系统中几毫安输出电流足以满足许多 CMOS 电路输入驱动的要求。

## 6.4.2 I/O 接口的应用

### 1. I/O 接口直接用于输入/输出

AT89S52 的 P0~P3 口直接用于输入/输出时, 都是准双向口。作为输出口时, 端口带有输出锁存器, 作为输入口时, 端口带有输入缓冲器, 但没有输入锁存器, 因此要输入的数据必须一直保持在引脚上, 直到把数据取走。AT89S52 的 P0~P3 口都有一定的带负载能力, 所以在有些简单应用的场合, 可以直接与开关、发光二极管等外部设备相连。

由于 AT89S52 的 I/O 接口只有数据口而没有状态口或控制口, 在实际使用时, 利用其 I/O 接口可以位寻址的特点, 特别是在查询式输入/输出传送时, 可以用 I/O 接口的某一位(或几位)来作为状态信息的传送者, 通过查询这一位的状态来确定外设是否处于“准备好”的状态。

**例 6-1** 为模拟图 6.4.1(a)的逻辑功能, 设计了图 6.4.1(b)单片机电路, 其中: 用 P1 口的 P1.0、P1.1 作为变量输入口, 用 P1.2 作为电路输出口, 并用一个发光二极管来显示



```

ORL      C,F          ;C←E∧F
ANL      C,/D         ;C←(E∧F)∧(E∨F)
MOV      P1.2,C       ;输出结果
SJMP     LOOP1        ;准备下一次模拟

```

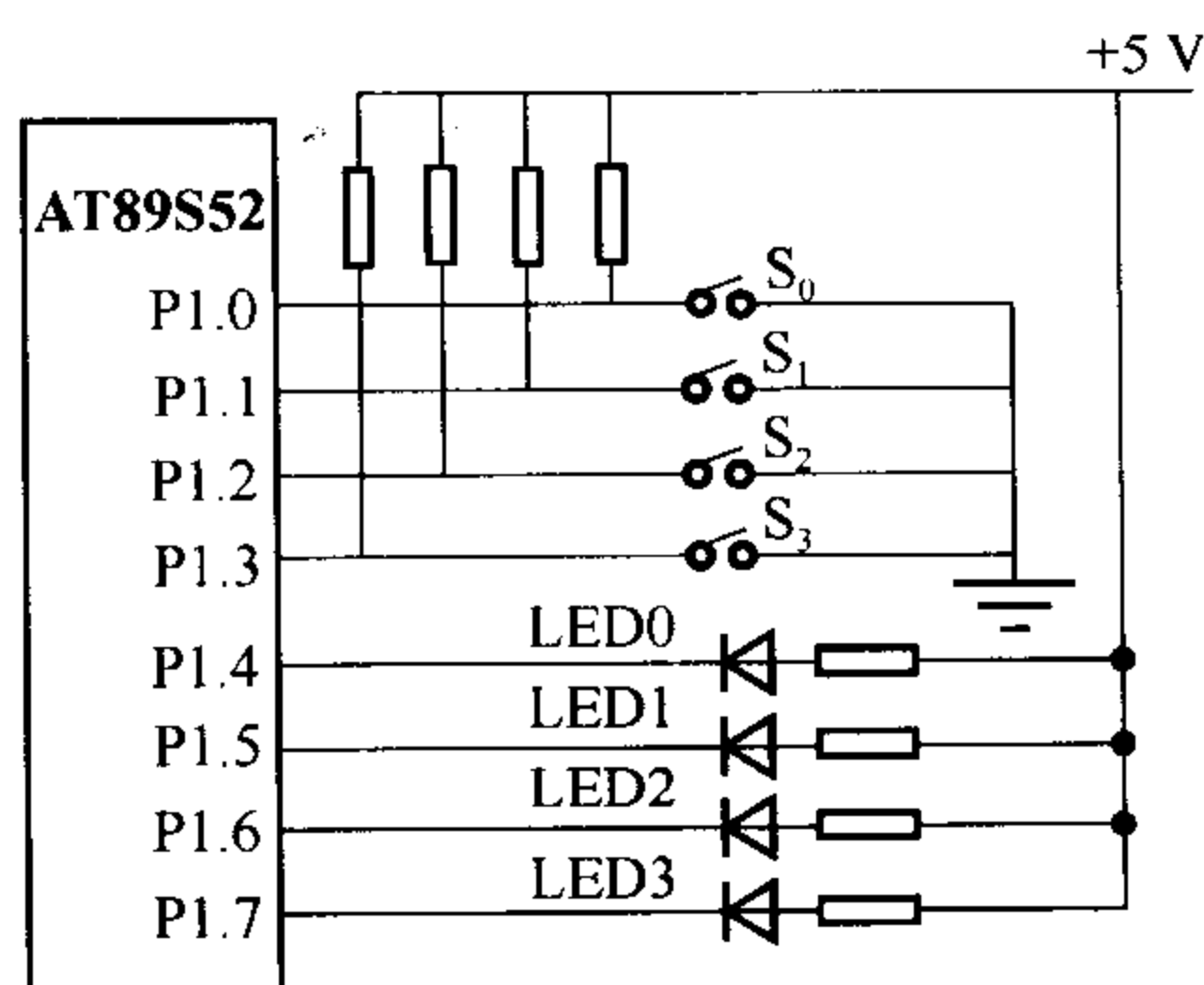


图 6.4.2 AT89S52 单片机与开关(键)、发光二极管的接口

本程序在读入引脚信号之前,都通过 ORL 指令使有关的位置 1,但不改变其他位的内容。

**例 6-2** 如图 6.4.2 所示为 AT89S52 单片机与开关(按键)、发光二极管的接口电路。单片机 P1.3~P1.0 连接到逻辑开关  $S_3 \sim S_0$ ; P1.7~P1.4 连接到发光二极管 LED3~LED0。编写程序,要求发光二极管 LED3~LED0 的亮、灭与开关  $S_3 \sim S_0$  的接通和断开状态相对应,当改变开关状态时,可观察到发光二极管的变化。试编写程序。

**解** 观察图中,可以分析,本例中单片机 P1.3~P1.0 作为数据输入口, P1.7~P1.4 作为输出口。开关状态输入显示程序清单如下。

```

LOOP: MOV      A, #0FH          ;P10~P13 口线送 1,作输入
      MOV      P1,A            ;
      MOV      A,P1            ;P1 口状态输入
      SWAP     A                ;开关状态到高 4 位
      MOV      P1,A            ;开关状态输出
      AJMP     LOOP            ;循环

```

## 2. P2 口总线/口线复用技术的实际应用

由于单片机本身资源有限,实际应用中经常需要扩展程序存储器、数据存储器及 I/O 接口等,单片机与外扩的这些资源之间的数据交流是通过外部总线来实现的。实际应用中, P0 口通常作为数据/低位地址总线复用; P1 口一般就作为通用 I/O 接口; P2 口则既可作为高位地址总线,又可作为一般 I/O 口线; P3 口常作为专用口。因此相对来说, P2 口的用法较为复杂。下面将以 AT89S52 为例结合实例详细说明单片机 P2 口的口线复用技术。

**例 6-3** 在某仪器的开发中遇到了这种情况:单片机 AT89S52 内部的存储器容量已经够用,需要连接液晶显示器、打印机、键盘等外部设备,其中液晶驱动器与单片机的接口要求有片选信号线 (LCD $\overline{CS}$ )、液晶复位信号线 (RST)、命令/数据控制线 (C/D)、读/写信号线 ( $\overline{RD}$ ,  $\overline{WR}$ ) 和 8 根数据线 (D0~D7);打印机的接口要求数据选通信号 (STB)、打印机忙检测信号 (BUSY) 以及 8 根数据线 (D0~D7);键盘需要 4 根线(输入)。P1 口已经被占用。



解 P1 口已经被占用, P0, P2 未外接存储器, 因此很自然地想到要利用 P2 口, 否则口线不足, 用 8255 等接口扩展芯片又过于浪费, 所以就采用了如图 6.4.3 所示的连线方式(P0 口接数据线, 图中略去未画, 其他无关的连线也省略未画)。

显然, 此处的 P2 口是作为一般 I/O 接口, 其中的 P2.7~P2.4 作为按键输入口, P2.3~P2.0 作为输出口。但是仔细分析就会发现, 液晶显示器需要片选信号, 其工作程序中仍要用到 MOVX 指令, 也就是说, 液晶显示器在这里相当于一个容量小于 256 B 的外部数据存储器。

很多人认为, 系统中没有外部程序存储器只有外部数据存储器, 而且容量在 256 B 以下时, 由于通过 P0 口送出 8 位地址就可以了, P2 口引脚上的数据不会在访问外部数据区时改变, 所以 P2 口仍可作通用 I/O 接口用。这种说法是不严密的, 实际是忽视了执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令和执行 MOVX @Ri, A 或 MOVX A, @Ri ( $i=0$  或 1) 指令对 P2 口的不同影响。

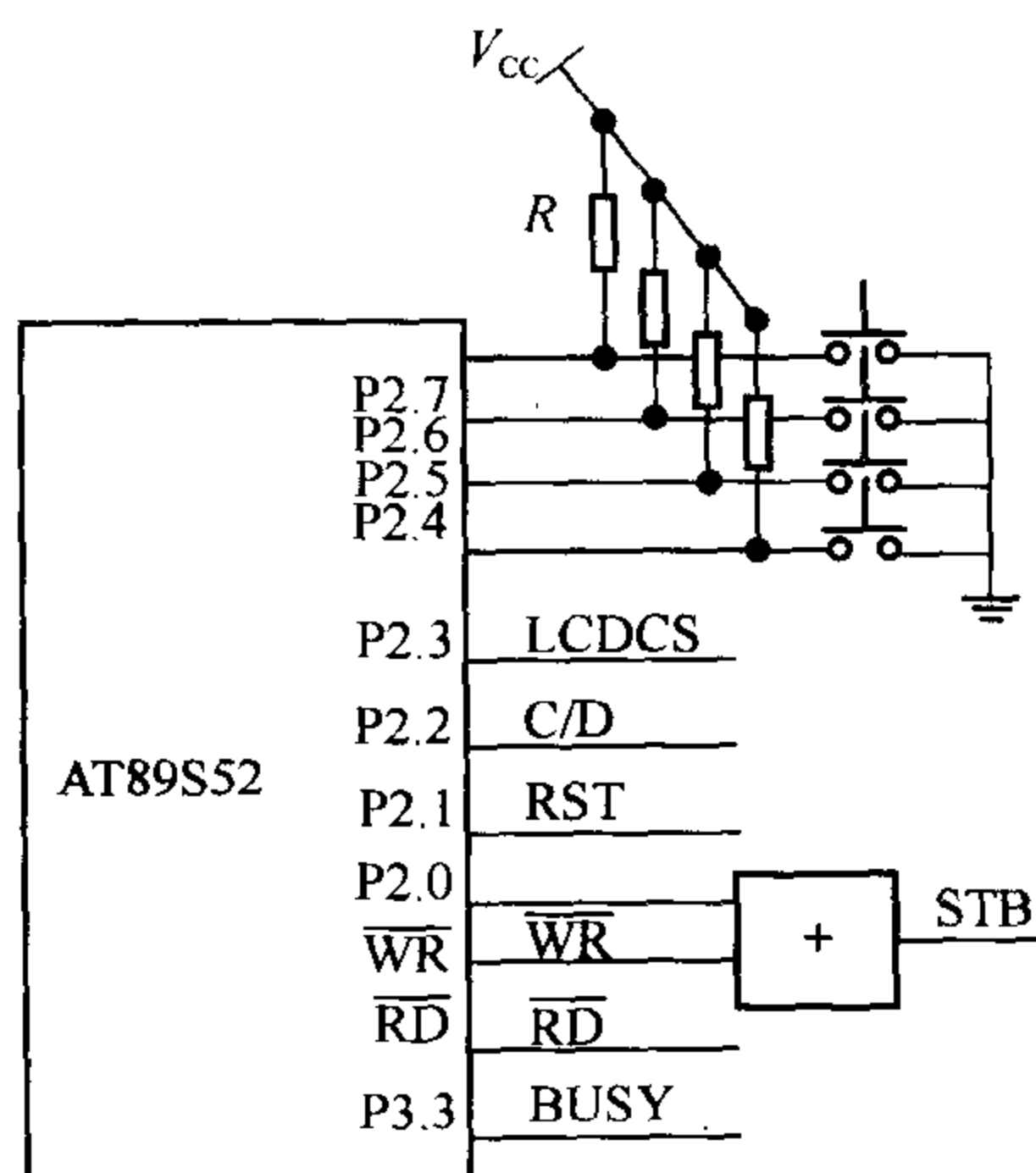


图 6.4.3 系统连线简图

图 6.4.4 和图 6.4.5 是访问外部数据存储器的时序图。由时序图可知, 无论外部数据存储器的容量多大, 在执行 MOVX 类指令读/写外部存储器时, P2 口都会出现“地址 A15~A8”。具体地说, 若执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令, P2 口引脚上会在指令执行期间出现 DP0H (DPTR 的高 8 位地址, 根据 AUXR1 中 DPS 的取值决定使用 DPTR0 或 DPTR1, 设 DPS=0) 值; 若执行 MOVX @Ri, A 或 MOVX A, @Ri 指令, P2 口引脚上则会出现特殊功能寄存器(口锁存器)P2 的值。

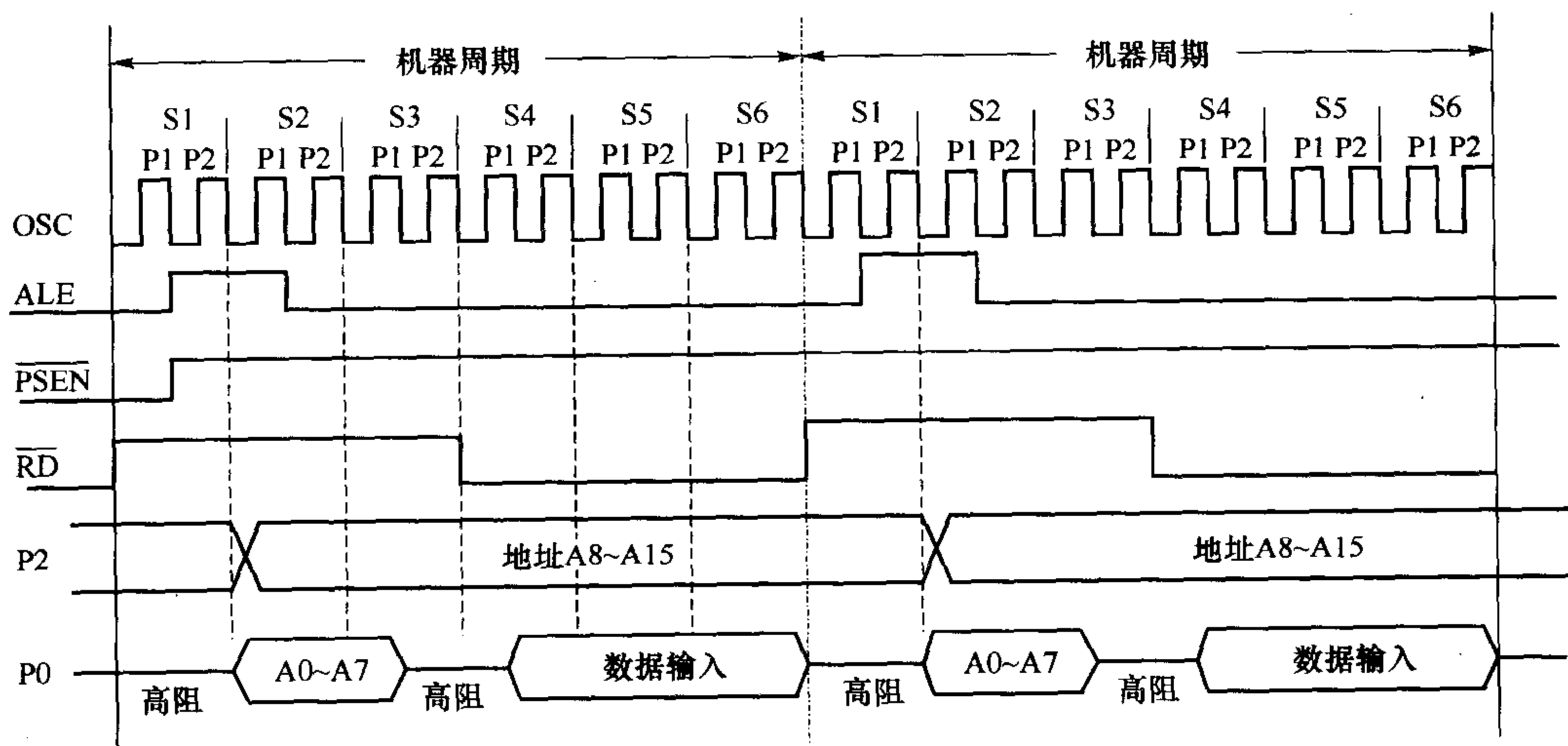


图 6.4.4 外部数据存储器读时序

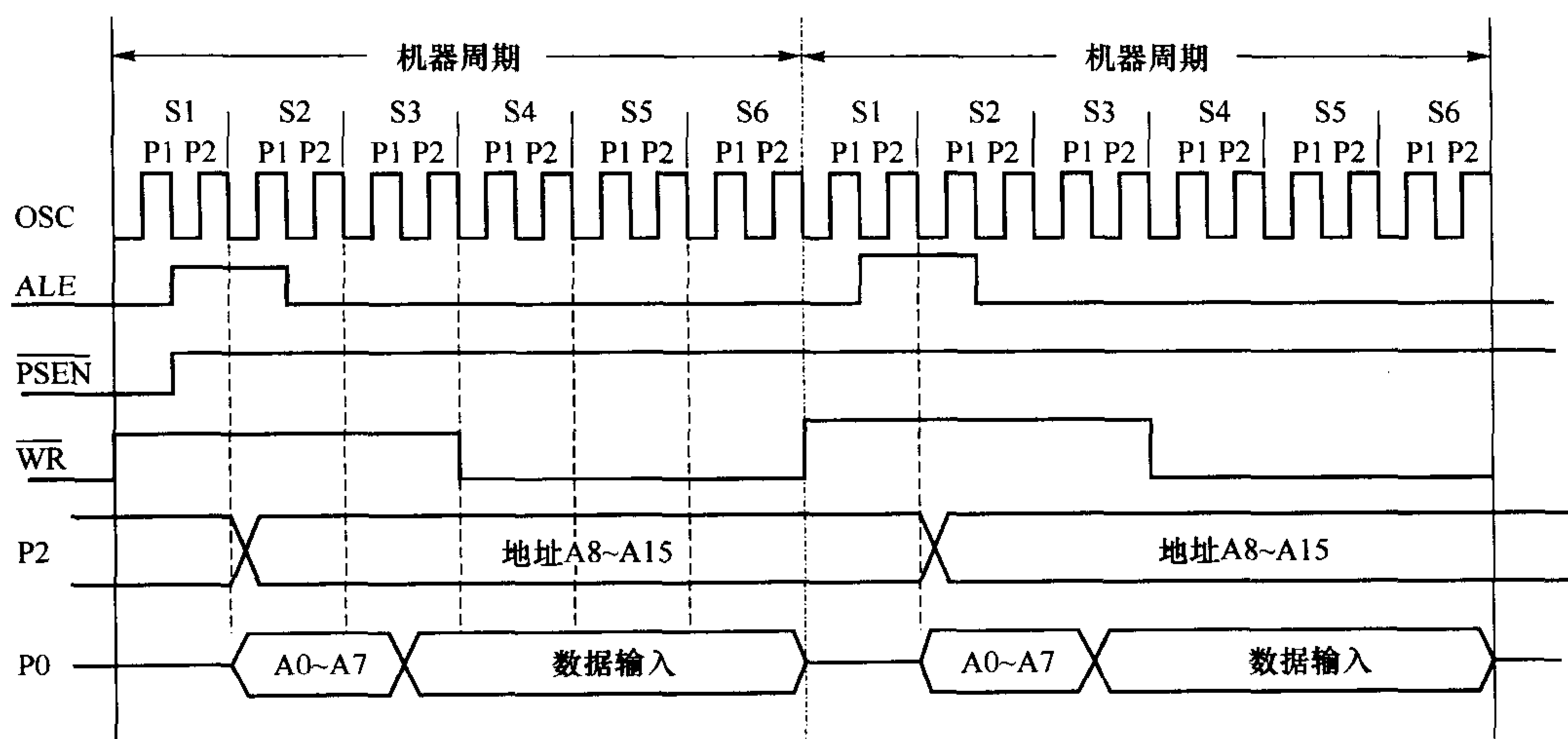


图 6.4.5 外部数据存储器写操作时序

那么,当外接数据存储器容量在 256 B 以下时,虽然只有通过 P0 口送出的 8 位地址对寻址有实际意义,但是 P2 口引脚上仍会出现 DP0H 或 P2 寄存器的值。若执行 MOVX @Ri, A 或 MOVX A, @Ri 指令,因为出现的是特殊功能寄存器(口锁存器)P2 的值,与执行指令前无异,可以不必考虑;若执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令,且数据指针高 8 位寄存器 DP0H 的值与特殊功能寄存器 P2 的值不一致,就会在执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令的瞬时改变 P2 口的状态。对于此时正作为一般 I/O 接口使用的 P2 口,则会出现如下两种情况。

(1) P2. × 作输出口,控制某个外部设备。例如控制某个发光二极管,按正常工作应使该位为 0,而 DP0H 中对应位却是 1,那么执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令,就可能使发光二极管瞬时发光或熄灭。这个时间很短,人眼也未必能够发觉,一般可以不予考虑;但是在某些场合却可能对系统安全或精度产生不利影响,不可掉以轻心。

(2) P2. × 作输入口,要先写入 1 再读入引脚上来自外设的状态,这正是准双向 I/O 接口的意义。如图 6.4.3 所示的 P2. 4~P2. 7 作为按键输入口,DP0H 对应的位可能是 0,那么执行 MOVX @DPTR, A 或 MOVX A, @DPTR 指令时,出现在 P2. × 引脚上的值就是 0,此时会出现一个很小的瞬时电流,但是执行完 MOVX @DPTR, A 或 MOVX A, @DPTR 指令后,P2. × 引脚上的值会恢复为 P2. × 端口内部输出锁存器的值,所以没有什么破坏性的影响,一般也不会影响按键的正常使用。

由以上分析可见,当外部数据存储器容量在 256 B 以下时,虽然只有通过 P0 口送出的 8 位地址对寻址有实际意义,但是把 P2 口作为一般 I/O 接口使用时却要区别对待:如果作为按键输入口就可以不予特殊考虑;如果作为输出口则要保证 P2 口引脚上的状态不被改变,也就是读/写外部数据存储器时 DP0H 的内容与 P2 寄存器无冲突,或者干脆在这种情况下不使用 MOVX @DPTR, A 或 MOVX A, @DPTR 指令读/写外部存储器,只使用 MOVX @Ri, A 或 MOVX A, @Ri 指令读/写外部存储器。很多人习惯于用

DPTR 寻址存储器,以便于在一定范围内依次寻址存储器(利用 INC DPTR 指令)。用 @Ri 寻址外设时(把外设看成存储器,容量不能超过 256 B),很容易忽略这种情况,要特别加以注意。

因此,在对图 6.4.3 所示系统调试程序时一定要注意这一点,对液晶显示器寻址时不能采用 MOVX @DPTR, A 或 MOVX A, @DPTR 指令,否则可能使液晶显示器不能正常工作。

**例 6-4** 将上面的例子稍加改变:单片机采用 AT89S52 (此时 P1 接口可以作为一般的双向 I/O 接口使用),要求外接一个 8 KB 的 RAM 6264,再连接液晶显示器、打印机、键盘等外部设备以及一个多路转换开关芯片(要 2 根控制线)。

**解** 外部数据存储器 6264 地址线的高 5 位为 P2.0~P2.4,片选线为 P2.5,实际地址空间可以是 0C000H~0DFFFH。只要 P2.5=0,就可以对 6264 的 8 KB 空间进行读/写。P2.7 和 P2.6 无论是 0 是 1 均不影响寻址,所以可以考虑利用 P2.7 和 P2.6。有如下几种可能的利用方式:

- 可以连接按键,不必担心执行 MOVX 类指令时 P2 口线会出现 0 或 1。
- 可以分别连接 LCDCS 和 C/D,把液晶显示器看成外部数据存储器,命令和数据分别占据不同的空间,只要不让液晶显示器和 RAM 同时选中即可。
- P2.6 和 P2.7 也可以作普通输出口(液晶显示器复位信号、多路转换开关芯片控制信号)使用,但要在软件上保证对外部数据存储器寻址时不改变输出口 P2.6 和 P2.7 的状态,如何保证呢?

如果作输出口用的位状态已知或不变(比如液晶显示器复位信号),则可以根据其是 0 还是 1 用指令把该位映射到高位地址 DP0H 或 P2 中。以对 P2.7 的操作为例,给 P2 或 DP0H 送完高位地址后,再执行指令 SET BP2.7 或 ORL DP0H, #80H 就可以令 P2.7 位在之后执行 MOVX 类指令对外读/写时仍保持 1 不变;同理,给 P2 或 DP0H 送完高位地址后,再执行指令 CLR P2.7 或 ANL DP0H, #7FH,就可以使 P2.7 位在之后执行 MOVX 类指令对外读/写时仍保持 0 不变。

当作输出口用的位状态未知或变化频繁时(例如多路转换开关的控制信号),若每次对外读/写时都先判断一下其是 0 是 1 再读写则过于麻烦。可以设置一个 P2 寄存器的暂存单元,借助该暂存单元把要隔离的输出口位保护起来,再恢复到对外读/写要用的高位地址中去。执行下面这两段程序的任何一个都可以把 P2.6 和 P2.7 口隔离起来(注:30H 就是暂存单元的位置, #××H 为适应工作要求使 P2.6 和 P2.7 为 0 或 1 的某立即数, #××××H 为任意存储器单元或外部设备的地址, #LLH 为任意存储器单元或外部设备的低位地址, #HHH 为任意存储器单元或外部设备的高位地址)。

程序 1:

```

MOV    P2, # × × H      ;给 P2 口送立即数使 P2.6 和 P2.7 作为输出口
MOV    30H, # × × H     ;正常工作,同时将此状态保存到暂存单元 30H 中
      ∴
ANL    30H, #11000000B   ;暂存单元中被保护位以外的位清 0
MOV    DPTR, # × × × H   ;送要访问的外部地址给 DPTR

```



```

ANL      DPOH, #00111111B ;将 DPOH 中与 P2 应被保护位对应的位清 0
ORL      DPOH, 30H        ;把 P2.6 和 P2.7 的状态恢复到 DPOH 中的相应位
MOVX     @DPTR, A          ;对外写,或 MOVX A, @DPTR 对外读
      ⋮

```

#### 程序 2:

```

MOV      P2, #××H          ;给 P2 口送某立即数使 P2.6 和 P2.7 作为输出口
MOV      30H, #××H         ;正常工作,同时将此状态保存到 30H 中
      ⋮
ANL      30H, #11000000B    ;把暂存单元中被保护位以外的位清 0
MOV      Ri, #LLH           ;送低位地址给 Ri
MOV      P2, #HHH           ;送高位地址给 P2 寄存器,P2.6 和 P2.7 可能为任意数
ANL      P2, #00111111B     ;P2.6 和 P2.7 位清 0
ORL      P2, 30H            ;把保存在 30H 中的 P2.6 和 P2.7 位的状态
                                ;回复到用于寻址的高位地址寄存器中的相应位
MOVX     @Ri, A              ;对外写,或 MOVX @Ri, A 对外读
      ⋮

```

只要改变上述程序中的某些常数,就可以实现对任意 P2.× 的隔离保护。这种解决方法同样也适用于只外接容量在 256 B 以下的数据存储器的情况,只是一般没有这个必要。

本节结合实例讨论了单片机 P2 口作为地址总线口和一般 I/O 接口的复用技术,得出如下结论:作为默认的地址总线口,即使不接地址线,在执行 MOVX 类指令时,P2 口的 8 根口线上依然会瞬时出现数据指针高 8 位 DP0H 或 P2 锁存器的值,因而要根据具体情况采取措施。如果单片机系统中外部数据存储器的容量不超过 256 B,把 P2 口当成一般 I/O 接口使用的最简单的做法是在读/写外部数据存储器或外设时仅仅使用 MOVX @Ri, A 或 MOVX A, @ Ri 指令;如果无外部程序存储器,而外部数据存储器容量超过 256 B,但 P2 口仍有部分口线对于寻址无影响,则可以不加特殊处理地把它们作为按键输入口使用,也可以通过软件方法把它们作为输出口使用,条件是保证寻址时不改变那些作输出口用的位状态。

## 习 题

1. 什么是 I/O 接口? I/O 接口的作用是什么?
2. AT89S52 单片机片内设有 4 个并行 I/O 接口,使用时有哪些特点和分工? 简述各个并行 I/O 接口的结构特点。
3. 何谓准双向 I/O 接口? 在使用中如何正确处理?
4. 何谓分时复用? 在什么情况下出现复用? 在硬件中应作何处理?
5. AT89S52 单片机的并行 I/O 接口信息有几种读取方法? 读-修改-写操作是针对 I/O 接口的哪一部分进行的? 有什么优点?

## 实 践 训 练

学完本章内容后,为掌握接口的相关知识、了解 I/O 接口技术的基本使用方法、掌握接口程序的编程方法,可按照下述步骤进行实践训练。

1. 用硬件连接好如例 6.4.1 所示的电路,并连接好复位电路、晶振电路和电源电路,模拟调试书中的程序,并将程序下载到单片机中,测试开关  $S_1$ 、 $S_2$  在不同状态时,  $L_1$  的状态。

2. 根据例 6.4.1 的电路,编写程序以 50 Hz 的频率循环点亮 LED 发光管,并能够通过开关  $S_1$ 、 $S_2$  调整 LED 的发光时间,按动开关  $S_1$ ,频率以 1 Hz 为基础增大;按动开关  $S_2$ ,频率以 1 Hz 为基础减少,观察发光管发光频率。循环点亮时间用延时程序实现。

3. 根据图 6.1 中电路,组成独立式键盘,编写程序,当对应的  $S_0 \sim S_7$  按键按下时,相应的发光管  $L_0 \sim L_7$  点亮。

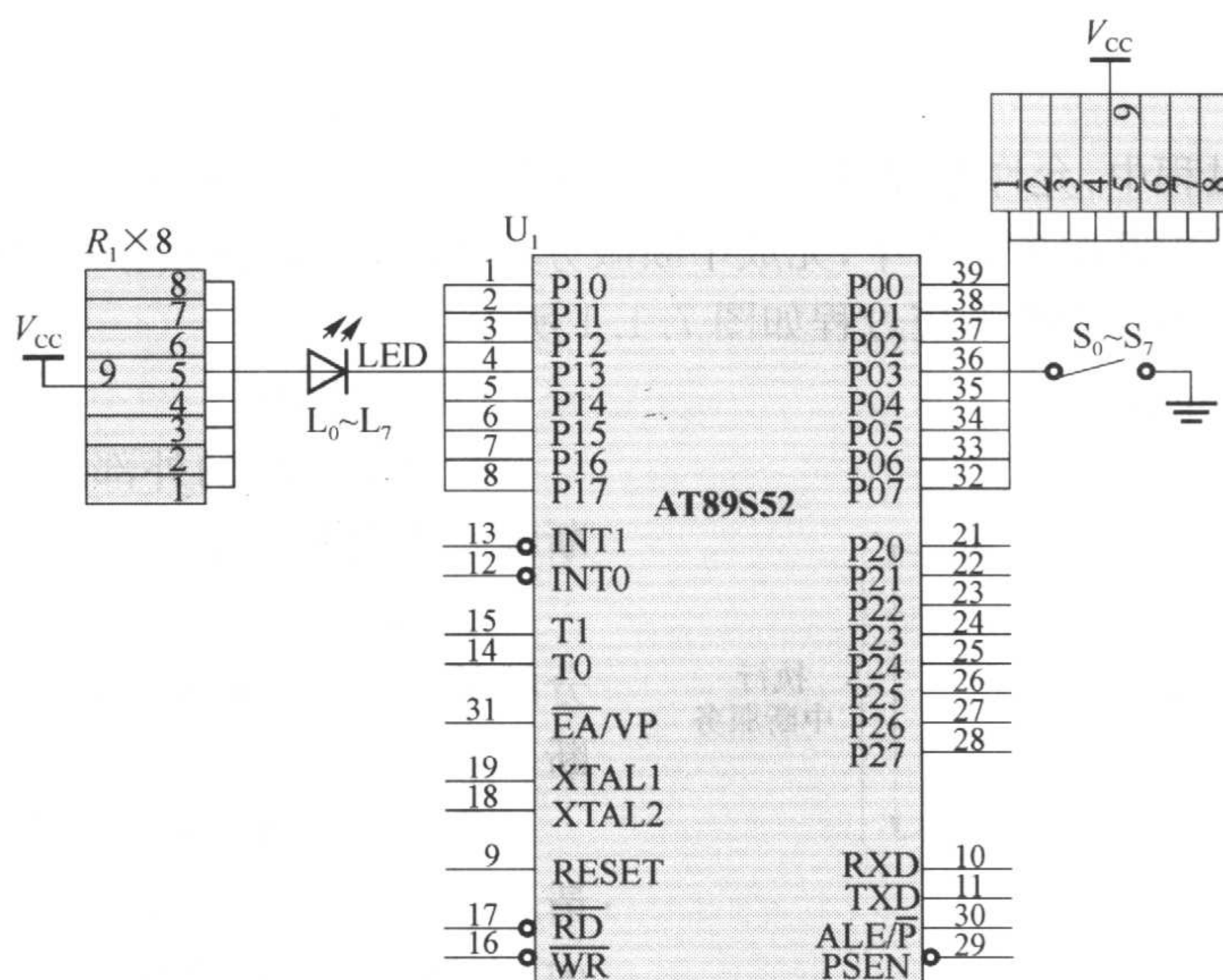


图 6.1 键盘显示电路

## 第 7 章 AT89S52 单片机中断系统

中断系统在单片机系统中起着十分重要的作用。一个功能很强的中断系统,能大大提高单片机处理事件的能力,提高效率,增强实时性。AT89S52 单片机的中断功能较强,设有 8 个中断源,共有 6 个中断矢量:定时/计数器 0,1,2,  $\overline{\text{INT0}}$ ,  $\overline{\text{INT1}}$  和一个串行通信中断矢量。有两级中断优先级,可实现两级中断嵌套。用户可以很方便地通过软件实现对中断的控制。本章介绍 AT89S52 单片机中断系统的结构及应用。

### 7.1 中断概述

#### 1. 中断系统的概念

##### (1) 中断

程序执行过程中,允许外部或内部事件通过硬件中断程序的执行,使其转向为处理外部或内部事件的中断服务程序中;完成中断服务程序后,CPU 继续原来被打断的程序,这样的过程称为中断。中断响应过程如图 7.1.1 所示。

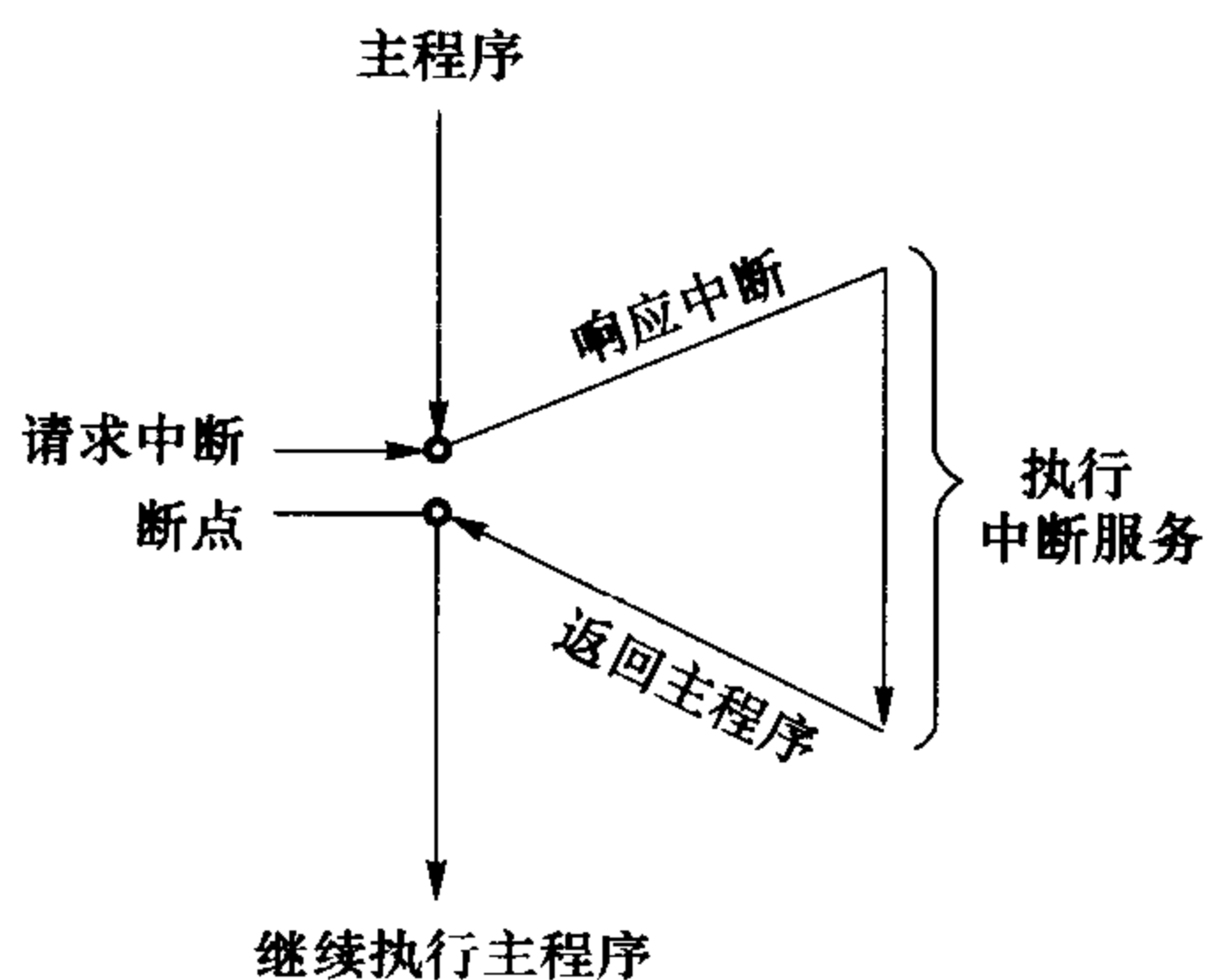


图 7.1.1 中断响应过程示意图

##### (2) 中断源

能产生中断的外部 and 内部事件称为中断源。中断源通常可分为以下几种。

① 设备中断:由单片机系统各组成部分的外围设备发出的中断申请,称为设备中断。如键盘、打印机、A/D 转换器等。

② 定时中断:定时时钟提出的中断申请。例如,在定时控制或定时数据采集系统中,由外部时钟电路定时,一旦到达规定的时间,时钟电路就向 CPU 发出中断申请。

③ 故障源中断:目前,单片机的 RAM 多采用半导体存储器,所以在电源掉电时,需要接入备用电源供电,以便保护存储器 RAM 中的信息。一般的做法是,在直流电源上并联电容,当电容电压因电源掉电下降到一定值时就发出中断申请,CPU 响应中断,执行保护现场信息的操作。

④ 程序性中断源:为调试程序而设置断点、单步工作等。

对于每个中断源,不仅要求能发出中断请求信号,而且这个信号还要能保持一定的时间,直至 CPU 响应这个中断请求后才能而且必须撤销这个中断请求信号。这样既不会因 CPU 未及时响应而丢失中断申请信号,也不会出现多次重复中断的情况。所以,要求每个中断源的接口电路中有一个中断请求触发器。另外,在实际系统中,往往有多个中断源,为了增加控制的灵活性,在每个中断源的接口电路中还设置一个中断屏蔽触发器,由

它控制该中断源的中断申请信号能否送到 CPU。

### (3) 中断优先级

随着中断技术的发展,在中断系统中设有多个中断源。当有两个以上的中断源同时请求中断时,CPU 应作何处理?这就提出了中断优先级问题,CPU 应响应优先级高的中断请求。不同的单片机中断系统各有不同的处理方法。除采用硬件中断优先级排队电路、优先权比较电路、按优先顺序查询等方法外,还可采用软件随机设定等方法。

### (4) 中断识别方式

单片机中断源的个数(AT89S52 设有 8 个中断源),反映了该单片机处理中断的能力。由于设有多个中断源,中断系统必须具备正确的识别中断的功能,才能可靠地为其服务。一般设有两种识别方式:查询中断和矢量中断。

查询中断方式是通过软件逐个查询各中断源的中断请求标志,其查询顺序反映了中断源的优先顺序。先查询的优先级高,后查询的优先级低。通过查询找出申请中断的中断源,然后转向相应的中断服务程序执行。其缺点是软件查询循环占用一定的时间,每次必须从优先级最高的中断查询开始,逐级向优先级低的中断查询,影响 CPU 响应中断的效率,而且优先级低的中断请求被响应的概率会受影响。

矢量中断方式(又称向量中断)则以硬件为基础,为每个中断源直接提供对应的中断服务程序入口地址,或称矢量地址。中断请求通过优先级排队电路,一旦被响应,立即转向对应的矢量地址去执行。因此,矢量中断响应速度快,提高了 CPU 的响应效率。

### (5) 中断的其他概念

中断请求:中断源中断信号有效时产生中断请求。

中断标志:伴随中断请求在 SFR 中登录的标记,如 IE0,IF0,TF1 等。

中断允许:允许某中断源中断响应的操作,也称开中断。

中断入口地址(中断矢量):指中断服务程序的入口地址。

中断嵌套:中断嵌套是对中断服务程序的中断操作。中断嵌套的中断源应比被嵌套的中断源有更高的优先级。

中断保护:中断保护是指中断响应后保护正在执行程序中使用的资源免遭中断服务程序的破坏。通常将需要保护的资源送入堆栈中,或利用更换工作寄存器区的方法实现。

中断服务程序:指中断响应后,中断源要求的处理操作程序。

中断源清除:中断响应后应撤销中断请求标志,以免中断返回后引发又一次中断。

中断屏蔽(禁止):屏蔽 CPU 中断(即禁止某些中断),中断系统中可用程序设置中断允许和中断屏蔽。

中断返回:中断返回是中断服务程序中的最后一条指令 RETI,执行该条指令后,程序返回原中断断点处的下一条指令处。

中断等待:中断等待是指中断允许(执行中断允许指令)后等待中断源出现中断请求。这种情况往往用于中断程序调试中。

## 2. 单片机中断系统需要解决的问题

单片机中断系统需要解决的问题主要有以下 3 点。

① 当单片机内部或外部有中断申请时,CPU 能及时响应中断,停止正在执行的任



务,转去处理中断服务子程序,中断服务程序执行完成后能回到原断点处继续处理原来的任务。

② 当有多个中断源同时申请中断时,应能先响应优先级高的中断源,实现中断优先级的控制。

③ 当低优先级中断源正在执行中断服务程序时,若这时优先级比它高的中断源也申请中断,要求能停止低优先级中断源的服务程序,转去执行更高优先级中断源的服务程序,实现中断嵌套,并能逐级正确返回原断点处。

### 3. 中断的主要功能

#### (1) 实现 CPU 与外部设备的速度协调

由于应用系统的许多外部设备速度较慢,可以通过中断的方法来协调快速 CPU 与慢速外部设备之间的工作。

#### (2) 实现实时控制

在单片机中,依靠中断技术能实现实时控制。实时控制要求单片机能及时完成被控对象随机提出的分析和计算任务。在自动控制系统中,要求各控制参量随机地在任何时刻可向单片机发出请求,CPU 必须快速响应、及时处理。

#### (3) 实现故障的及时发现及处理

单片机应用中由于外界的干扰以及硬件或软件设计中存在问题等因素,在实际运行中会出现硬件故障、运算错误、程序运行故障等,有了中断技术,单片机就能及时发现故障并自动处理。

#### (4) 实现人机联系

比如通过键盘向单片机发出中断请求,可以实时干预单片机的工作。

由于中断技术的采用,推动了单片机科学和技术的发展,拓宽了单片机的应用以及各应用领域的自动化和智能化水平。

### 4. 实际应用中需注意的问题

(1) 由于中断的发生是随机的,因而使得由中断驱动的中断服务程序难于把握、检测和调试,这就要求在设计中断和中断服务程序时应特别谨慎,力求正确。

(2) 在输入输出的数据处理频度很高或实时处理的要求很高时,不宜采用中断方式,因为由中断源请求中断到 CPU 响应中断需要一定的时间。

## 7.2 中断系统结构与中断控制

AT89S52 的中断系统共有 8 个中断源,6 个中断矢量,两级中断优先级,可实现两级嵌套,可通过软件来屏蔽或允许相应的中断请求。

### 7.2.1 AT89S52 的中断源

图 7.2.1 为 AT89S52 单片机的中断功能示意图。外部中断源  $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$  对应两个中断矢量;串行通信有接收中断源和发送中断源,经过一个或门,共用同一个中断矢量;定时/计数器 0、定时/计数器 1 的溢出中断源对应两个中断矢量;定时/计数器 2 有计数溢

出和捕获两种中断源,经或门共用一个中断矢量。

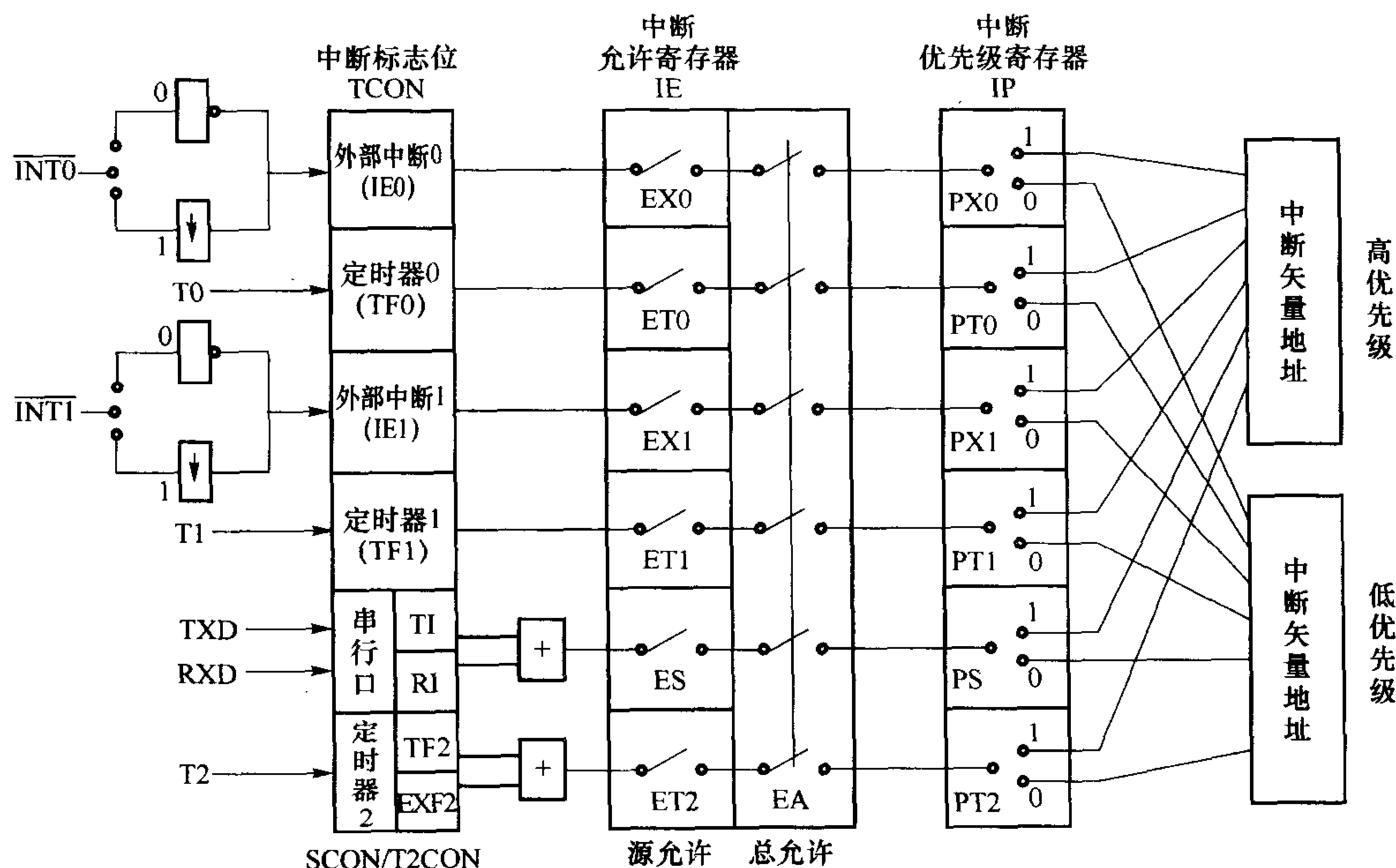


图 7.2.1 AT89S52 的中断系统结构示意图

$\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$ 的中断请求信号由外部产生并输入,称外部中断,其余的中断请求信号均由 CPU 内部产生,故称为内部中断。各中断请求信号分别由定时/计数器控制寄存器 TCON、定时/计数器 2 控制寄存器 T2CON 和串行通信控制寄存器 SCON 的相应位锁存,提供给 CPU 查询和采样。

### 1. 外部中断源

$\overline{\text{INT0}}$ :外部中断 0 请求中断输入引脚(P 3.2 引脚),低电平或下降沿(从高到低)有效。 $\overline{\text{INT0}}$ 中断请求有效,则置位 IE0(TCON.1)中断请求标志位。CPU 在每个机器周期的 S5P2 状态采样 IE0 标志位,当条件满足,则响应中断请求。中断响应后,转向对应的中断矢量,执行中断服务程序,并由硬件自动复位 IE0 标志位。当关中断时亦可采用软件查询法进行处理。

$\overline{\text{INT1}}$ :外部中断 1 请求中断输入引脚(P 3.3 引脚),低电平或下降沿(从高到低)有效。其功能与操作同 $\overline{\text{INT0}}$ 。

### 2. 内部中断源

定时/计数器 0 溢出中断:定时/计数器 0 无论内部定时或对外部事件 T0 计数,当计数器(TH0,TL0)计数溢出置位 TF0(TCON.5)中断请求标志位。CPU 在每个机器周期的 S5P2 状态时采样 TF0 标志位,当条件满足时 CPU 响应中断请求,转向对应的中断矢量,执行该中断服务程序,并由硬件自动将 TF0 标志位清 0。

定时/计数器 1 溢出中断:其功能和操作类似定时/计数器 0。其中断请求标志位为 TCON.7 的 TF1。

定时/计数器 2 溢出中断: AT89S52 的定时/计数器 2 有两个中断源, 有两种不同的工作方式。

- 定时/计数器方式。当定时/计数器方式的计数器 (TH2, TL2) 计数满后溢出, 置位中断请求标志位 TF2 (T2CON. 7), 向 CPU 请求中断处理。
- “捕获”方式。当外部输入端口 T2EX 发生从 1→0 下降沿时, 亦将置位中断请求标志位 EXF2 (T2CON. 6), 向 CPU 请求中断处理。

这两种中断请求, 在满足中断响应条件时, CPU 都将响应其中断请求, 转入同一个中断矢量地址进行中断处理。因此, 必须在该中断服务程序中进行 TF2 和 EXF2 两种中断请求标志位的查询、判断, 然后正确转入对应的中断服务程序。中断响应后, 必须用软件复位 TF2 或 EXF2 标志位。

串行通信中断。当完成一串行帧的接收/发送时, 分别置位串行通信控制寄存器 SCON 中的 RI/TI 中断请求标志位, 当条件满足时, CPU 响应中断请求。由于串行通信中的接收/发送 (RI/TI) 中断请求合用同一个中断矢量地址, 故必须在中断服务程序中查询、判断是接收 (RI) 还是发送 (TI) 请求中断处理, 然后转入对应的处理程序。无论采用中断或非中断方式, 均需用软件复位 RI/TI。

上述各中断均可通过软件对其中断请求标志位进行置位/复位, 这同内部硬件自动置位/复位的效果一样, 亦可通过软件产生中断请求或将挂起的中断请求撤销, 即所谓的软件中断。在关中断的情况下, 可采用软件查询方式进行处理。

### 3. 外部中断源触发方式

AT89S52 的两个外部中断源 ( $\overline{\text{INT0}}$ ,  $\overline{\text{INT1}}$ ), 有电平触发和下降沿触发两种方式, 通过软件编程可对两种中断请求触发方式进行选择。

#### (1) 电平触发方式

通过软件编程, 对中断控制寄存器 TCON 中的  $\text{IT}\times$  ( $\times$  为 0 或 1) 位设置为 0 ( $(\text{IT}\times)=0$ ) 时, 即选择  $\overline{\text{INT}\times}$  为电平触发方式。CPU 在每个机器周期的 S5P2 检测中断请求输入信号端口  $\overline{\text{INT}\times}$ , 若为低电平, 则外部中断请求有效, 置位 TCON 寄存器中的外部中断请求标志位  $\text{IE}\times$ , 向 CPU 申请中断。

对电平触发方式的外部中断源, 其中断请求信号 (低电平) 应持续保持请求有效, 直至 CPU 响应该中断请求为止, 这是因为中断系统对中断请求不作记忆。CPU 响应该中断后必须在该中断服务程序返回前撤销原中断请求 (使  $\overline{\text{INT}\times}$  变为高电平), 以避免再次进入该中断服务程序。为保证中断请求能被正确采样,  $\overline{\text{INT}\times}$  端口的中断请求信号 (低电平) 至少应保持两个机器周期。

#### (2) 下降沿触发方式

通过软件设置 TCON 寄存器中的  $\text{IT}\times$  为 1 时, 则选择外部中断 ( $\overline{\text{INT}\times}$ ) 为下降沿触发方式。

设置为下降沿触发中断触发方式后, CPU 在相继两个机器周期进行检测, 前一个机器周期在  $\overline{\text{INT}\times}$  端口检测到高电平, 紧接的后一个机器周期检测到低电平, 则置位 TCON 寄存器中的  $\text{IE}\times$  中断请求标志位, 向 CPU 申请中断。

为保证中断请求能被正确采样, 中断请求信号至少应保持一个机器周期的高电平、一

个机器周期的低电平。

无论电平触发还是下降沿触发方式,一旦 CPU 响应中断,转向中断服务程序执行时,由内部硬件自动复位 TCON 寄存器中的中断请求标志位  $IE \times$  为 0。

## 7.2.2 中断标志与控制

单片机设置了 5 个专用寄存器用于中断控制,需正确设置其状态以便保证中断系统正常工作。5 个专用寄存器包括:定时/计数器 0、1 控制寄存器 TCON,定时器 2 控制寄存器 T2CON,串行口控制寄存器 SCON,中断允许控制寄存器 IE,中断优先级控制寄存器 IP。

$\overline{INT0}$ ,  $\overline{INT1}$ , T0 及 T1 的中断标志存放在 TCON(定时/计数器控制)寄存器中,串行口的中断标志存放在 SCON(串行口控制)寄存器中;T2 的中断标志存放在 T2CON(定时/计数器 2 控制)寄存器中。IP 用于定义各中断源的中断优先级,IE 用于确定各中断是允许还是禁止。

### 1. 定时/计数器 0、1 控制寄存器 TCON

TCON 是定时/计数器和外部中断两者合用的一个特殊功能寄存器,复位时为 00H,可位寻址。TCON 的字节地址为 88H,其格式如下。

	D7	D6	D5	D4	D3	D2	D1	D0	
TCON	TF1	—	TF0	—	IE1	IT1	IE0	IT0	字节地址: 88H
位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H	可位寻址

各位定义如下。

TF1(TCON. 7):定时/计数器 1 溢出中断请求标志位。当定时/计数器 1 计数产生溢出信号时,由内部硬件置位 TF1(TF1=1),向 CPU 请求中断,当 CPU 响应中断并转向该中断服务程序执行时,由内部硬件自动将 TF1 清 0。

TF0(TCON. 5):定时/计数器 0 溢出中断请求标志位。当定时/计数器 0 计数产生溢出信号时,由内部硬件置位 TF0(TF0=1),向 CPU 请求中断,当 CPU 响应中断并转向该中断服务程序执行时,由内部硬件自动将 TF0 清 0。

IE1(TCON. 3):外部中断( $\overline{INT1}$ )中断请求标志位。当 CPU 检测到 $\overline{INT1}$ 低电平或下降沿且(IT1)=1 时,由内部硬件置位 IE1 标志位(IE1=1),向 CPU 请求中断,当 CPU 响应中断并转向该中断服务程序执行时,由内部硬件自动清 0 IE1 标志位。

IE0(TCON. 1):外部中断( $\overline{INT0}$ )中断请求标志位。当 CPU 检测到 $\overline{INT0}$ 低电平或下降沿且(IT0)=1 时,由内部硬件置位 IE0 标志位(IE0=1),向 CPU 请求中断,当 CPU 响应中断并转向该中断服务程序执行时,由内部硬件自动清 0 IE0 标志位。

IT1(TCON. 2):用软件置位/复位 IT1 来选择外部中断 $\overline{INT1}$ 是下降沿触发还是电平触发中断请求。当 IT1 置 1 时,则外部中断 $\overline{INT1}$ 为下降沿触发中断请求,即 $\overline{INT1}$ 端口由前一个机器周期的高电平,跳变为下一个机器周期的低电平,则触发中断请求;当 IT1 复位清 0,则 $\overline{INT1}$ 的低电平触发中断请求。

IT0(TCON. 0):由软件置位/复位 IT0 位来选择外部中断 $\overline{INT0}$ 是下降沿触发还是低



电平触发中断请求,其控制原理同上。

## 2. 定时器 2 控制寄存器 T2CON

定时器 2 控制寄存器 T2CON 字节地址为 C8H,其格式如下。

	D7	D6	D5	D4	D3	D2	D1	D0	
T2CON	TF2	EXF2	—	—	—	—	—	—	字节地址: C8H
位地址	CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H	可位寻址

TF2(T2CON. 7):定时/计数器 2 溢出中断请求标志位。当定时/计数器 2 计数产生溢出信号时,由内部硬件置位 TF2,向 CPU 请求中断,必须由软件复位。

EXF2(T2CON. 6):定时/计数器 2 外部中断请求标志位。若由引脚 T2EX(P1. 1)上的下降沿引起“捕获”或“重新再装入”且 EXEN2 位为 1,则置位标志位 EXF2,向 CPU 请求中断,中断响应后,必须由软件复位。

## 3. 串行口控制寄存器 SCON

串行口控制寄存器 SCON 字节地址为 98H,其格式如下。

	D7	D6	D5	D4	D3	D2	D1	D0	
SCON	—	—	—	—	—	—	TI	RI	字节地址: 98H
位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H	可位寻址

SCON 低两位是接收中断源 RI 和发送中断源 TI 标志。

TI(SCON. 1):串行口发送中断标志。发送完一帧,由硬件置位。中断响应后,必须用软件清 0。

RI(SCON. 0):串行口接收中断标志。接收完一帧,由硬件置位。中断响应后,必须用软件清 0。

## 4. 中断允许控制寄存器 IE

AT89S52 的中断均属可屏蔽中断,即通过软件对特殊功能寄存器 IE 的设置,实现对各中断源的中断请求允许(开放)或屏蔽(禁止)的控制。中断控制寄存器 IE 的格式及各位含义如下。

	D7	D6	D5	D4	D3	D2	D1	D0	
IE	EA	—	ET2	ES	ET1	EX1	ET0	EX0	字节地址: A8H
位地址	AFH	9EH	ADH	ACH	ABH	AAH	A9H	A8H	可位寻址

EA(IE. 7):全部中断允许/禁止位。若(EA)=0,则禁止响应所有中断;若(EA)=1,则各中断源响应与否取决于各自的中断控制位的状态。

(IE. 6):保留位,无意义。

ET2(IE. 5):定时/计数器 2 溢出或捕获中断响应控制位。若(ET2)=0,则禁止中断响应;若(ET2)=1,则允许中断响应。

ES(IE. 4):串行通信接收/发送中断响应控制位。若(ES)=0,禁止中断响应;若

(ES)=1,允许中断响应。

ET1(IE.3):定时/计数器1溢出中断响应控制位。若(ET1)=0,禁止中断响应;若(ET1)=1,允许中断响应。

EX1(IE.2):外部中断 $\overline{\text{INT1}}$ 中断响应控制位。若(EX1)=0,禁止中断响应;若(EX1)=1,允许中断响应。

ET0(IE.1):定时/计数器0溢出中断响应控制位。若(ET0)=0,禁止中断响应;若(ET0)=1,允许中断响应。

EX0(IE.0):外部中断 $\overline{\text{INT0}}$ 中断响应控制位。若(EX0)=0,禁止中断响应;若(EX0)=1,允许中断响应。

从上可见 AT89S52 的中断响应为两级控制,EA 为总的中断响应控制位,每一个中断源还有各自的中断响应控制位。

### 5. 中断优先级控制寄存器 IP

AT89S52 的中断设有两级优先级,每个中断源均可通过软件对中断优先级寄存器 IP 中的对应位进行设置,编程为高优先级或低优先级,置 1 为高优先级,清 0 为低优先级。正在执行的低优先级中断服务程序可以被高优先级中断源的中断请求所中断,但不能被同级的或低优先级的中断源的中断请求所中断;正在执行的高优先级的中断服务程序不能被任何中断源的中断请求所中断。两个或两个以上的中断源同时请求中断时,CPU 只响应优先级高的中断请求。为了实现上述规则,中断系统内部设有两个不可寻址的中断优先级状态触发器,其中一个用于指示正在服务于高优先级的中断,并阻止所有其他中断请求的响应,另一个则用于指示正在服务于低优先级的中断,除能被高优先级中断请求所中断外,阻止被其他同级或低级的中断请求所中断。

中断优先级控制寄存器 IP 的字节地址为 B8H,具有位寻址功能,可通过软件设置各个中断源的中断优先级。IP 控制寄存器的格式如下。

	D7	D6	D5	D4	D3	D2	D1	D0	
IP	—	—	PT2	PS	PT1	PX1	PT0	PX0	字节地址: B8H
位地址	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H	可位寻址

(IP.6、IP.7):保留位,无定义。

PT2(IP.5):定时/计数器2的中断优先级设置位。通过编程置位(PT2)=1,定义为高优先级;清0(PT2)=0,则定义为低优先级中断。

PS(IP.4):串行通信中断优先级设置位。软件置位(PS)=1,则定义为高优先级中断;清0(PS)=0,则定义为低优先级中断。

PT1(IP.3):定时/计数器1中断优先级设置位。具体设置和含义同上。

PX1(IP.2):外部中断 $\overline{\text{INT1}}$ 中断优先级设置位。具体设置和含义同上。

PT0(IP.1):定时/计数器0中断优先级设置位。具体设置和含义同上。

PX0(IP.0):外部中断 $\overline{\text{INT0}}$ 中断优先级设置位。具体设置和含义同上。

复位后 IP 的内容为 00H。具体应用时应将系统中实时性强和要求及时性的中断源设置为高优先级。

当同时有两个或两个以上相同优先级的中断请求时,则由内部按查询顺序决定优先响应的中断请求,排在顺序后面的、未被响应的中断请求将被挂起。其优先顺序由高向低顺序排列。优先顺序排列如表 7.2.1 所示。

表 7.2.1 中断优先顺序

顺 序	中断请求标志	中断源名	优先顺序
1	IE0	外部中断 0 (INT0)	<div style="text-align: center;">           最高            ↓            最低         </div>
2	TF0	定时/计数器 0 溢出中断	
3	IE1	外部中断 1 (INT1)	
4	TF1	定时/计数器 1 溢出中断	
5	RI+TI	串行通信中断	
6	TF2+EXF2	定时/计数器 2 溢出中断	

这种同级内的中断优先顺序仅用来确认多个(两个或两个以上)同级中断源同时请求中断时的优先响应顺序,而不能中断正在执行的同一优先级的中断服务程序。

以上所述可归纳为如下基本规则:

- (1) 任一中断源的高或低优先级中断均可通过软件对 IP 的相应位进行设置;
- (2) 不同优先级中断源同时请求中断时,优先响应高优先级的中断请求,高优先级中断请求可中断正在执行的低优先级的服务程序,实现两级嵌套,同级或低优先级的中断请求不能实现中断嵌套;
- (3) 同一优先级的多个中断源同时请求中断时按表 7.2.1 的优先顺序查询确定,优先响应顺序高的中断。

## 7.3 中断响应

### 7.3.1 中断响应条件

为保证正在执行的程序不因随机出现的中断响应而被破坏或出错,又能正确保护和恢复现场,必须对中断响应提出要求。

CPU 响应中断首先是中断源有中断请求,而且是在允许中断响应(即 IE 寄存器中的 (EA)=1 且对应的控制位置 1)的条件下,CPU 在每个机器周期的 S5P2 状态采样中断请求标志位,在下一个机器周期对采样到的中断请求按中断优先级或优先顺序进行处理,对确认的中断请求必须满足下列条件:

- (1) 无同级或高优先级中断服务程序正在执行中;
- (2) 当前指令已执行到最后一个机器周期并已结束;
- (3) 当前正在执行的不是返回 (RET, RETI) 指令或访问 IE 和 IP 特殊功能寄存器指令。

满足上述 3 个条件,则中断系统由硬件生成一条内部长调用 (LCALL) 指令,控制程序转向对应的中断服务程序去执行。只要上述 3 个条件中有一条不满足,就将丢弃本次

中断请求而取消中断响应。

上述3个条件中的第一条保证正在执行的同级或高优先级的中断服务不被中断；第二条保证正在执行的当前指令执行完成而不被破坏；第三条除保证正在执行的返回指令或访问 IE 和 IP 指令执行完外，还必须执行完紧跟其后的下一条指令，以保证子程序的正确返回以及 IE 和 IP 寄存器功能的正确设置和有效。

### 7.3.2 中断响应过程

图 7.3.1 为由中断源发出中断请求至 CPU 响应中断的时间顺序示意图。

#### 1. 中断采样

中断采样是针对外部中断请求信号进行的，而内部中断请求都发生在芯片内部，可以直接置位 TCON 或 SCON 中的中断请求标志。在每个机器周期的 S5P2 期间，各中断标志采样相应的中断源，并置入相应标志。这里需注意的是，如中断请求标志已被置位，但因前述的响应中断的3个条件不能满足而未被响应，待到封锁条件已撤除，该中断请求标志已不复存在（标志位已回0），则被拖延的中断请求就不再被响应（请求丢失）。也就是说，AT89S52 的中断系统对未被响应的中断请求标志（被置位状态）不作记忆。每个查询周期仅对前一个周期采样到的中断请求标志已置位状态进行中断响应处理。为此，未被及时响应的中断请求有可能丢失。

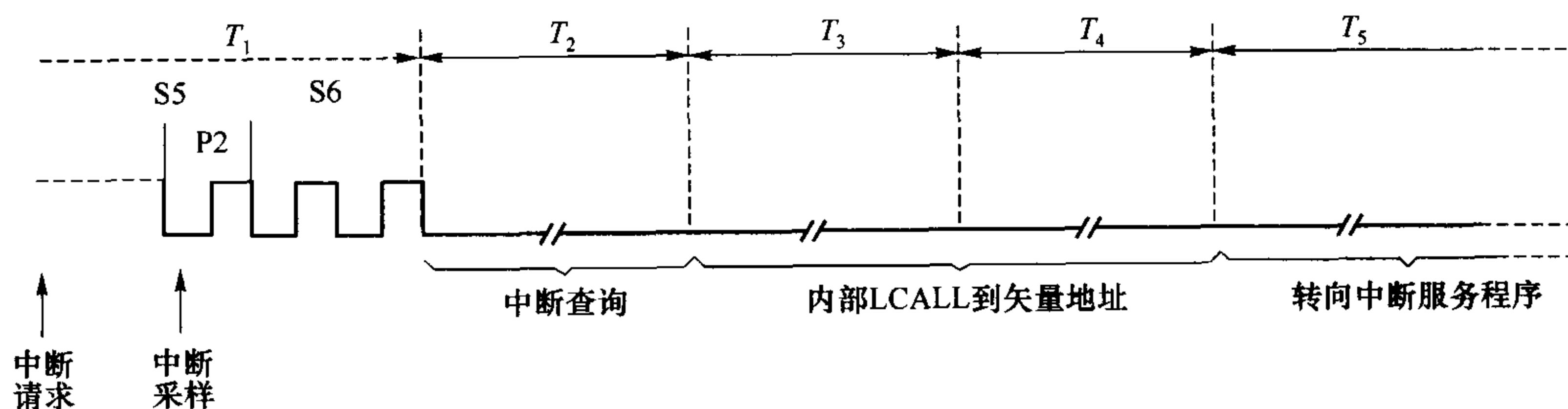


图 7.3.1 响应中断的时间顺序示意图

图中  $T_1 \sim T_5$  为机器周期， $T_1$  表示在这个机器周期中某中断源请求中断，并置位 TCON 寄存器中相应的中断请求标志位，在该周期的 S5P2 状态被 CPU 采样。在随后的  $T_2$  机器周期，执行中断优先级处理。

#### 2. 中断查询

若查询到某中断标志为 1，则按优先级的高低进行处理，即响应中断。

AT89S52 的中断请求都汇集在 TCON、T2CON 和 SCON 3 个特殊功能寄存器中。而 CPU 则在下一机器周期的 S6 期间按优先级的顺序查询各中断标志，先查询高级中断，再查询低级中断，同级中断按内部中断优先级序列查询。如果查询到有中断标志位为 1，则表明有中断请求发生，接着从相邻的下一个机器周期的 S1 状态开始进行中断响应。

由于中断请求是随机发生的，CPU 无法预先得知，因此中断查询要在指令执行的每个机器周期中不停地重复执行。

#### 3. 中断响应

如当前执行的指令不是返回或访问 IE 和 IP 指令，而是其他指令的最后机器周期，且

中断请求有效(即响应中断请求),则进入  $T_3$ 、 $T_4$  机器周期,由硬件产生内部长调用(LCALL)指令,且置位优先级状态触发器,将断点地址(PC 的当前值)压入堆栈保护,把对应的中断矢量地址送 PC,同时清 0 该中断请求标志位(但不能清除串行通信或定时/计数器 2 的有关中断请求标志位,它们必须由软件另行清 0),从而控制程序转向该中断服务程序去执行。

CPU 中断处理从响应中断、控制程序转向对应的中断矢量地址入口处执行中断服务程序,直到执行返回(RETI)指令为止。RETI 指令的执行,一方面告知中断控制系统,中断服务程序已执行完毕,清除中断优先级状态触发器;另一方面将原压入堆栈的断点地址(PC 值)弹出装入程序计数器 PC 中,从而返回原程序的断点处继续往下执行。其他断点信息应由软件实现保护和恢复。

AT89S52 的指令系统中设有两条返回指令:RET 和 RETI。子程序返回用 RET 指令,中断服务程序返回用 RETI 指令。如采用的是 RET 返回指令,虽然也能使中断服务程序返回原断点处继续往下执行原程序,但它不会告知中断控制系统,现行中断服务程序已执行完毕,致使中断控制系统误认为仍在执行中断服务程序而屏蔽新的中断请求。因此,中断服务程序的返回必须用 RETI 指令,而不能用 RET 返回指令代替。

可以看出,中断的执行过程与调用子程序有许多相似点:

- (1) 都是中断当前正在执行的程序,转去执行子程序或中断服务程序;
- (2) 都是由硬件自动地把断点地址压入堆栈,然后通过软件完成现场保护;
- (3) 执行完子程序或中断服务程序后,都要通过软件完成现场恢复,并通过执行返回指令,重新返回到断点处,继续往下执行程序;
- (4) 二者都可以实现嵌套,如中断嵌套和子程序嵌套。

但是中断的执行与调用子程序也有一些差别:

- (1) 中断请求信号可以由外部设备发出,是随机的,如故障产生的中断请求,比如按键中断等,子程序调用却是由软件编排好的;
- (2) 中断响应后由固定的矢量地址转入中断服务程序,而子程序地址由软件设定;
- (3) 中断响应是受控的,其响应时间会受一些因素的影响,而子程序响应时间是固定的。

### 7.3.3 中断响应时间

从中断源发出中断请求,到 CPU 响应中断请求转向对应的中断服务程序开始执行所需要的时间,称为中断响应时间,如图 7.3.1 所示。中断源发出中断请求信号有效,置位相应的中断请求标志位,每个机器周期的 S5P2 状态被采样并锁存,接着在下一个机器周期进行中断优先查询。CPU 响应中断可能有以下几种情况。

- (1) 经中断优先查询,如果中断请求有效且满足中断响应的 3 个条件,则 CPU 响应中断请求,转向对应的中断矢量地址为入口的中断服务程序开始执行。

长调用(LCALL)是条双周期指令。因此,中断系统从中断采样,经中断优先查询及生成和执行 LCALL 指令,共需 3 个机器周期,才开始执行中断服务程序。这是最快的中断响应,其中断响应时间为 3 个机器周期。

- (2) 中断请求发生在指令的开始周期(双周期或 4 周期指令),或是第一个机器周期

的开始,而 CPU 采样有效、响应中断请求必须在指令的最后机器周期的 S5P2 状态。因此,待这个指令执行完就需 2~4 个机器周期,在这种情况下,中断响应需 5~7 个机器周期。

(3) 当中断请求发生在正在执行的返回指令(RET、RETI)或访问 IE、IP 指令周期,则在本指令执行完后,还需执行完紧接其后的一条指令才被响应。这时中断响应延迟时间不会超过 5 个机器周期。这样,总的中断响应时间一般不超过 8 个机器周期。

(4) 如果该中断请求遇到高优先级或同级中断服务程序正在执行中,则本次中断请求将被挂起。

可见,从中断源发出中断请求到 CPU 响应中断、转去执行中断服务程序需 3~8 个机器周期。这就可能给高要求的应用造成延误,请在实际应用中加以注意。

中断技术在单片机应用中越来越显示出其重要性,特别是在满足实时性要求时尤为重要。一般单片机中设置的中断源有限,在实际应用中应该巧妙安排,认真设计和编写中断服务程序,尽力确保其正确性,以免给程序调试带来麻烦。

## 7.4 中断请求的撤除

CPU 响应中断请求,转向执行中断服务程序,在其执行中断返回指令(RETI)之前,中断请求信号必须撤除,否则将会再一次引起中断而出错。中断请求撤除的方式有如下 3 种。

### 1. 单片机内部硬件自动清除中断标志

对于定时/计数器 T0、T1 的溢出中断和采用下降沿触发方式的外部中断请求,在 CPU 响应中断后,由内部硬件自动清除中断标志 TF0 和 TF1、IE0 和 IE1,而自动撤除中断请求,即硬件置位,硬件清除。

### 2. 应用软件清除相应标志

对于串行接收/发送中断请求和定时/计数器 T2 的溢出和捕获中断请求,在 CPU 响应中断后,必须在中断服务程序中应用软件清除 RI, TI, TF2 和 EXF2 这些中断标志,才能撤除中断,即硬件置位,软件清除。

### 3. 采用外加硬件结合软件清除中断请求

对于采用电平触发方式的外部中断请求,中断标志的撤销是自动的,但中断请求信号的低电平可能继续存在,在以后机器周期采样时又会把已清 0 的 IE0、IE1 标志重新置 1,再次申请中断。为保证在 CPU 中断响应后、执行返回指令前,撤除中断请求,必须考虑另外的措施,保证在中断响应后把中断请求信号从低电平强制改变为高电平。可在系统中加入如图 7.4.1 所示的电路。

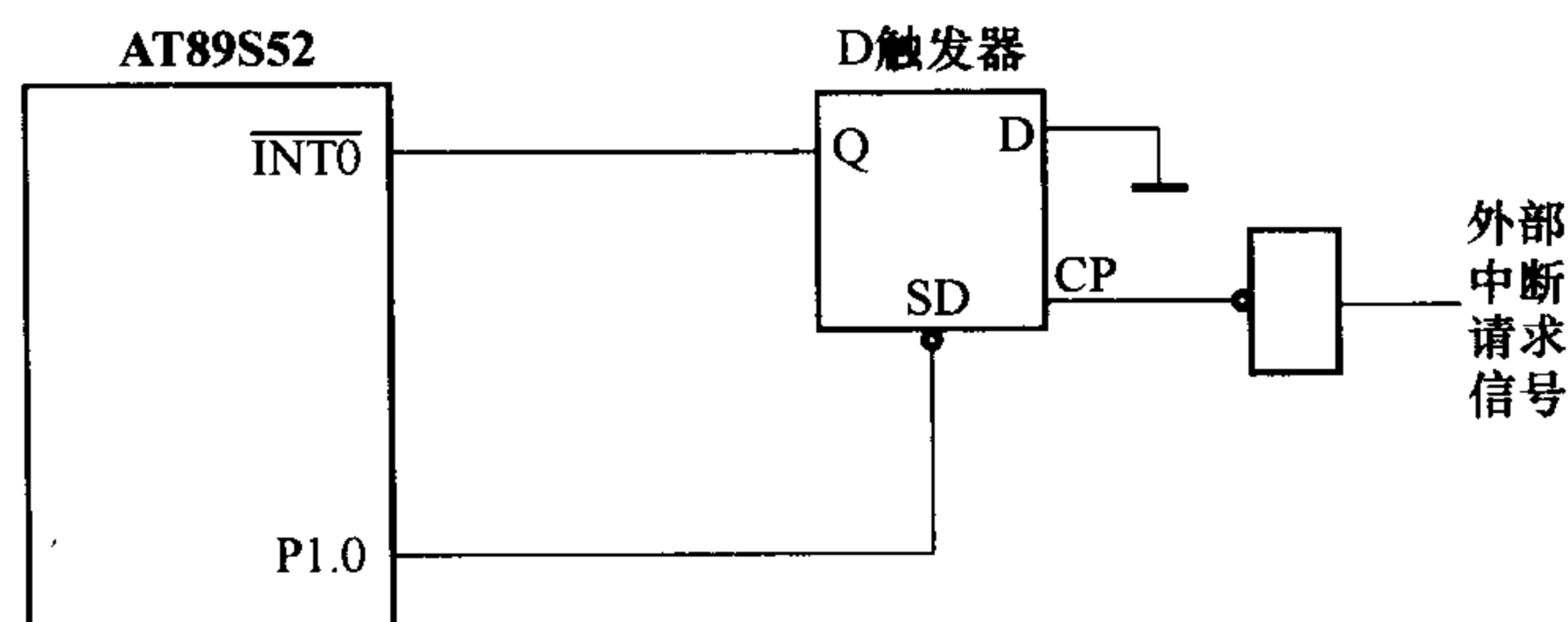


图 7.4.1 电平方式外部中断请求的撤除电路

从图 7.4.1 中可看到,用 D 触发器锁存外部中断请求低电平,并通过触发器输出端 Q 送 $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ ,所以 D 触发器对外部中断请求没有影响。但在中断响应后,为了撤销低电平引起的中断请求,可利用 D 触发器的直接置位端 SD 来实现。采用 AT89S52 的一根 I/O 口线来控制 SD 端。只要在 SD 端输入一个负脉冲(P1.0 初始状态为 1),即可使 D 触发器置 1,从而撤销了低电平的中断请求信号(硬件置位,硬、软件结合清除)。

所需负脉冲可以通过在中断服务程序中增加以下两条指令得到:

```
ANL    P1, #0FEH    ;Q 置 1(SD 为直接置位端,低电平有效)
ORL    P1, #01H     ;SD 无效
```

使 P1.0 输出一个负脉冲,其持续时间为两个机器周期,足以使 D 触发器置位,撤除低电平中断请求。第二条指令是必要的,否则 D 触发器的 Q 端始终输出 1,无法再接收外部中断请求。

## 7.5 外部中断源的扩展

在 AT89S52 系列单片机中,一般只有两个外部中断请求输入端 $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ 。当某个系统需要多个外部中断源时,可以通过增加 OC 门结合软件来扩展;当定时/计数器在系统中有空余时,也可以通过对计数器计数长度的巧妙设置,使定时/计数器的外部输入脚(T0, T1 或 T2)成为外部中断请求输入端。

### 7.5.1 采用“OC 门”经“线或”扩展中断源

利用芯片本身的外部中断请求输入端( $\overline{\text{INT0}}$ 和 $\overline{\text{INT1}}$ )就可很方便地扩展多个外部中断源。图 7.5.1 就是用 AT89S52 的一个外部中断源 $\overline{\text{INT0}}$ (或 $\overline{\text{INT1}}$ )扩展 4 个外部中断源的电路。

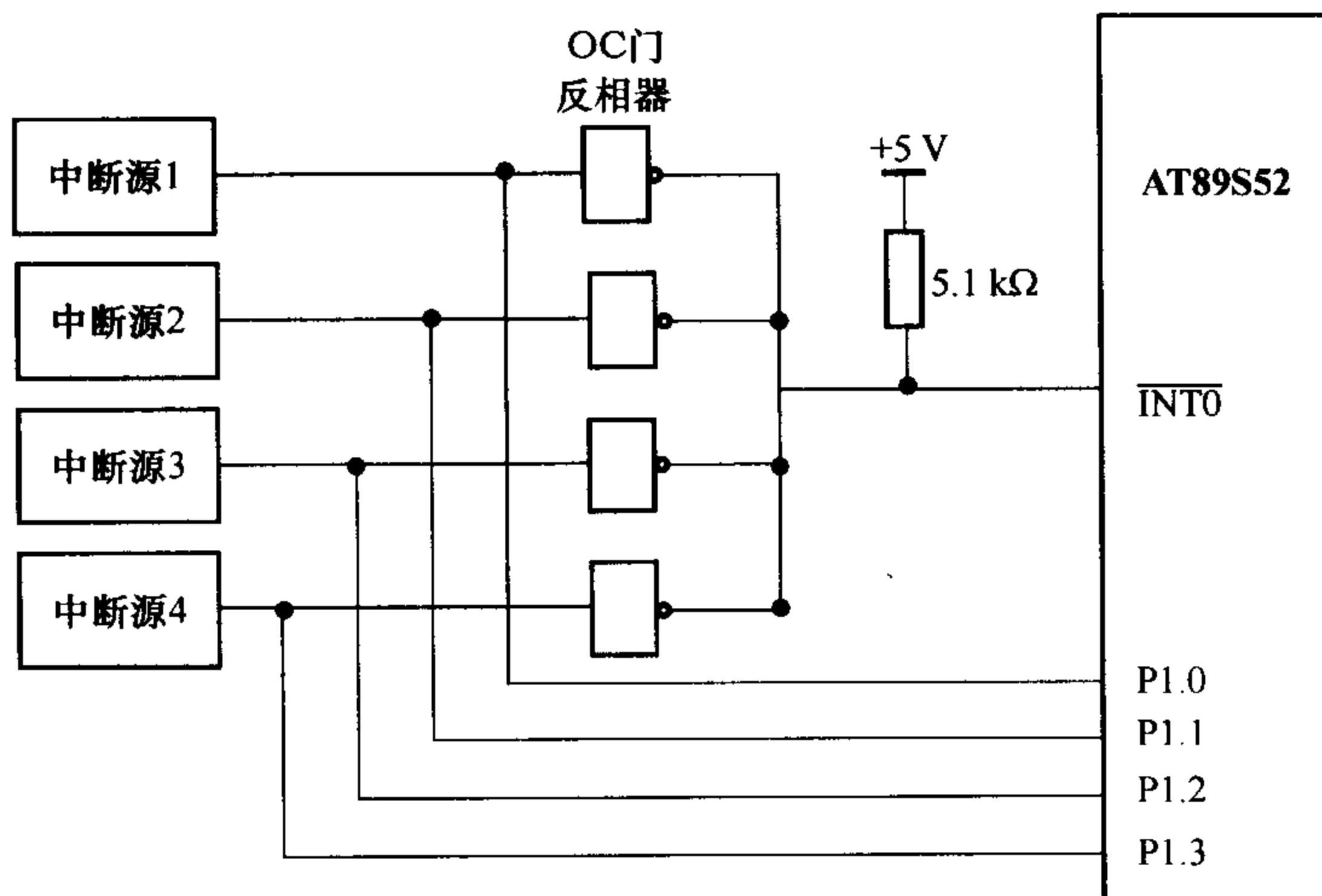


图 7.5.1 外部中断源扩展电路



4 个扩展外部中断源中有一个或几个出现高电平,则反相器输出为 0,引起 $\overline{\text{INT0}}$ 低电平触发中断,所以这些中断源都是电平触发方式。当满足外部中断请求条件时,CPU 响应中断,转入 0003H 单元开始执行中断服务程序。在中断服务程序中,由软件设定的顺序查询外中断哪一位是高电平,然后进入该中断处理程序。查询的顺序就是外部扩展中断源的中断优先级顺序。

外部中断源查询的流程图如图 7.5.2 所示。

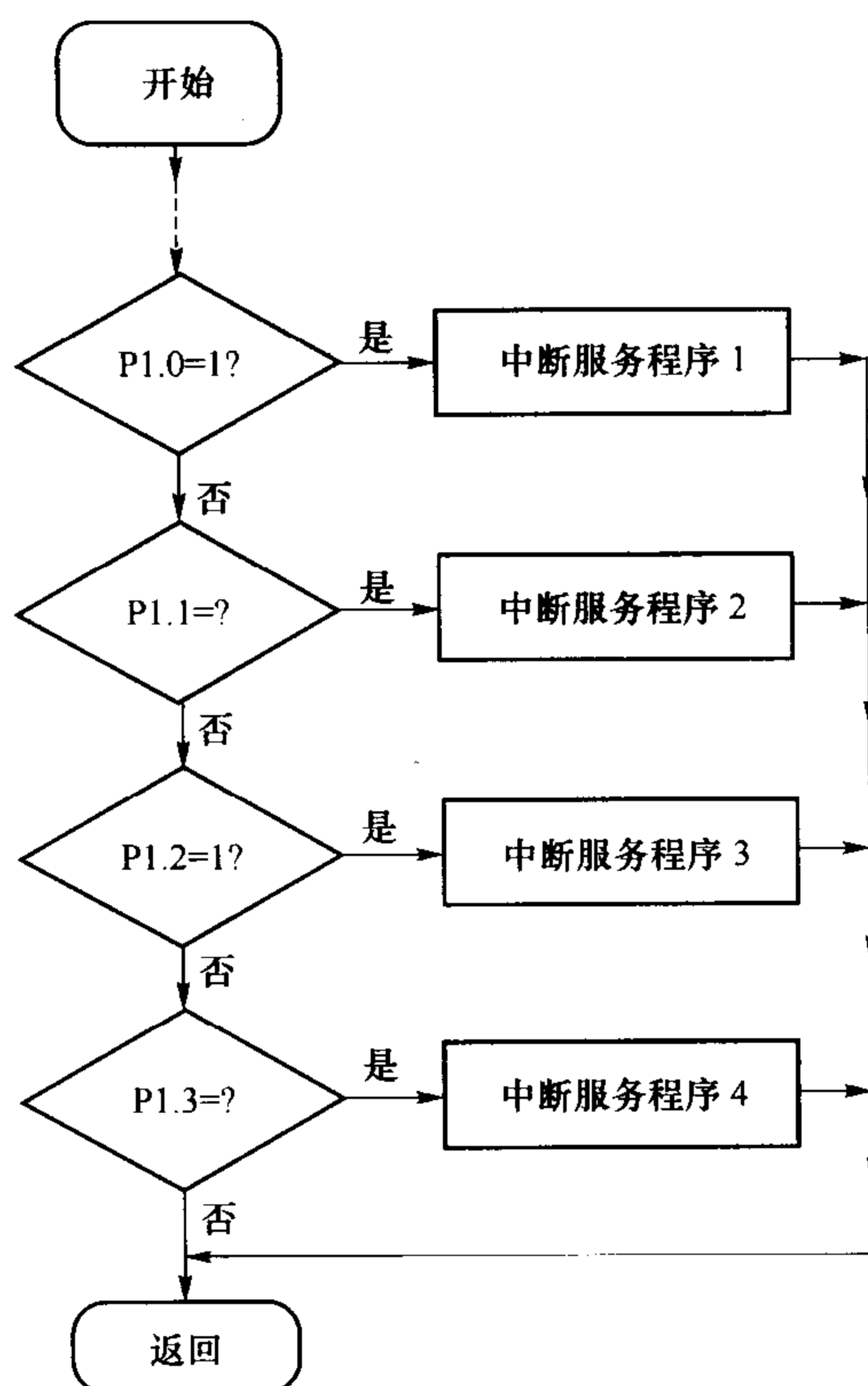


图 7.5.2 外部中断源查询的流程图

$\overline{\text{INT0}}$ 的中断服务程序如下:

```

PINT0:  PUSH    PSW                ;保护现场
        PUSH    ACC
        JB      P1.0, LOOP1        ;转向中断服务程序 1
        JB      P1.1, LOOP2        ;转向中断服务程序 2
        JB      P1.2, LOOP3        ;转向中断服务程序 3
        JB      P1.3, LOOP4        ;转向中断服务程序 4
INTEND:  POP     ACC                ;恢复现场
        POP     PSW
        RETI
LOOP1:   ...                      ;中断服务程序 1
  
```

```

                AJMP    INTEND
LOOP2:  ...                ; 中断服务程序 2
                AJMP    INTEND
LOOP3:  ...                ; 中断服务程序 3
                AJMP    INTEND
LOOP4:  ...                ; 中断服务程序 4
                AJMP    INTEND

```

### 7.5.2 通过片内定时/计数器扩展中断源

利用 T0 或 T1 的外部事件输入引脚 T0、T1 作为边沿触发的外部中断源。这时应设置定时/计数器为计数器方式,而计数常数为满刻度值。外部输入的脉冲在下降沿时有效,计数器加 1 后即溢出,向 CPU 申请中断。

如果以 T0、T1 的计数脉冲输入作为外部中断请求输入,T0、T1 的中断矢量作为第三、第四个扩展的外部中断矢量,T0、T1 的中断服务程序入口地址作为扩展的外部中断服务入口地址,即实现了外部中断的扩展。

当 T2 作为波特率发生器时,若 EXEN2 置 1,则 T2EX 端的信号产生下降沿时,EXF2 将置 1,但不会发生重装载或捕获操作。这时,T2EX 可以作为一个附加的外部中断源。

**例 7-1** 把外部中断请求信号 2 连到 T1 引脚上,定时/计数器 1 设为方式 2,即 8 位自动重装载方式,时间常数设为满刻度值 FFH。外部中断的服务程序入口地址存放在 T1 的中断矢量区中。其初始化程序段如下:

```

                ORG      0000H
                AJMP     MAIN
                ORG      001BH          ; T1 中断矢量作为外部中断 2 的中断矢量
                LJMP     INT2
                ORG      0030H
MAIN:  MOV     TMOD, # 60H          ; 设 T1 计数器方式 2
        MOV     TL1, # 0FFH        ; 置 T1 计数常数
        MOV     TH1, # 0FFH
        SETB    EA                  ; 开中断
        SETB    ET1                 ; 允许计数器 1 中断
        SETB    TR1                 ; 启动计数
        :
INT2:   :                          ; 外部中断 2 服务程序

```

## 7.6 中断程序设计

中断程序一般都包含两个部分,即主程序中的中断初始化和实现中断操作任务的中

断服务程序,如图 7.6.1 所示。在主程序中的任何地点都可设中断初始化,但只有在中断初始化开中断之后,有中断源请求中断时,才响应中断,将程序立即转移到该中断源的入口地址处,进入中断服务操作。中断服务操作结束后,程序又返回到主程序的中断断点处,继续执行原来被中断的主程序。

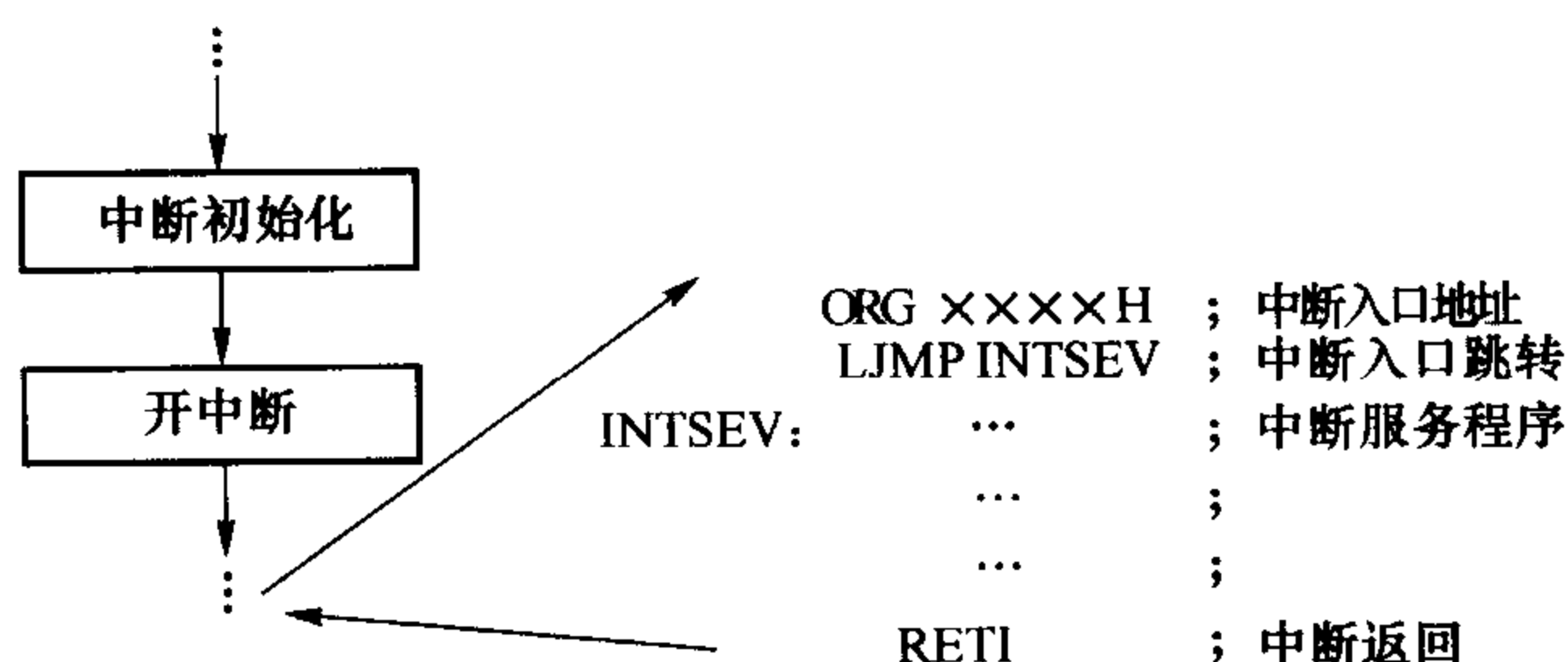


图 7.6.1 中断程序的一般格式

### 1. 主程序中的中断初始化

主程序中的中断初始化包括中断系统初始状态的设置、开中断和中断服务程序的前期初始化等。

#### (1) 初始状态设置

中断初始状态设置主要是对中断系统特殊功能寄存器进行设置来保证中断系统的工作状态,如设置中断源的优先级、设置中断源的触发方式等。如果中断初始化设置前中断系统状态已满足要求,可不再进行初始化设置。

#### (2) 开中断

开中断是对中断控制寄存器 IE 的设置。开中断是主程序中断初始化中唯一不可缺少的指令设置。只有开中断后,才能响应中断请求。

#### (3) 中断服务程序的前期初始化

中断服务程序中有一些需要在主程序中完成的初始化操作,如对定时器的初始化及计数初值的装载等。

### 2. 中断响应与中断服务程序

#### (1) 中断响应

主程序开中断后可随时响应中断。中断响应时,主程序立即中断当前运行的程序,断点地址自动压入堆栈,程序转入中断入口地址处。

#### (2) 中断转移

中断入口地址处设置一个中断服务程序的转移指令,将程序转移到中断服务程序的入口处,这样,中断服务程序可安放在程序存储器的任何空间。

#### (3) 中断服务程序

中断服务程序是中断响应后的主要操作内容,是一段完整的中断请求功能操作。一般情况下,在完成功能操作后便返回主程序。但考虑中断系统的运行特点,为提高在某些状态下系统运行的可靠性,在中断服务程序中要进行主程序资源保护、中断源清除和关中断等操作。

① 主程序的资源保护。由于中断响应是对主程序的随机插入性操作,在主程序断点前后资源必须连续使用,而该资源又可能被中断服务程序占用,必须将主程序中的资源压入堆栈保护,待中断返回前弹出堆栈。

② 中断源的清除。对于串行接收/发送中断请求和定时/计数器 T2 的溢出和捕获中断请求,在 CPU 响应中断后,必须在中断服务程序中应用软件清除 RI, TI, TF2 和 EXF2 这些中断标志,才能撤除中断(硬件置位,软件清除)。

③ 关中断。对于只需要一次中断服务操作的中断服务程序,中断返回前可设关中断操作指令。

#### (4) 中断返回

中断返回是任何中断服务程序都必须具备的最后一条操作指令 RETI, 执行到 RETI 时,主程序中的中断断点地址自动装入 PC 指针,程序便自动返回主程序,从中断断点后继续运行。正常情况下,中断服务程序中压入堆栈的数据在中断返回前必须如数退出,以保证断点地址在堆栈顶部,中断返回时能准确返回到中断响应的断点处。

### 3. 中断程序设计实例

**例 7-2** 编写外部按键输入的中断操作演示程序。要求:按图 7.6.2 所示电路,根据 S0, S1 按键的状态,点亮 L0, L1。按下 S0 点亮 L0 片刻,按下 S1 后点亮 L1 片刻。

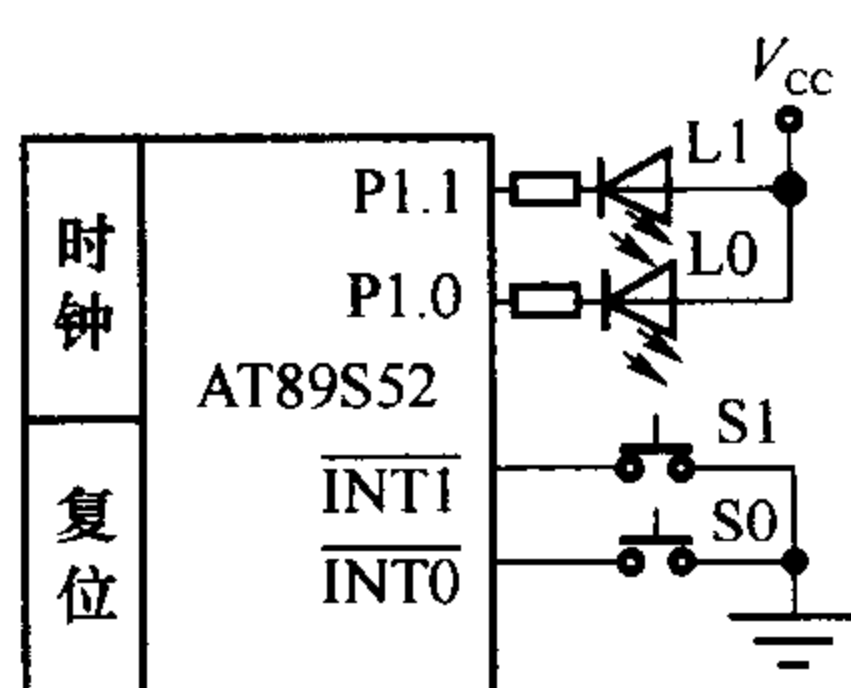


图 7.6.2 键盘中断电路图

#### 解 (1) 设计思路

这是一个两路外部中断输入演示电路,按下 S0, S1 都会立即中断原来的操作来点亮 L0 或 L1。中断初始化包括:

- ① 保证 L0, L1 为熄灭状态。
- ② 设置  $\overline{\text{INT0}}$ ,  $\overline{\text{INT1}}$  的触发方式。根据按键输入信号特点,选电平触发方式。
- ③ 设中断优先级。假定  $\overline{\text{INT0}}$ ,  $\overline{\text{INT1}}$  都设为低优先级。

#### (2) 中断应用程序设计包括主程序设计和中断服务程序设计

主程序清单如下:

```

ORG      0000H
LJMP     MKL
ORG      0003H      ;  $\overline{\text{INT0}}$  中断入口地址
LJMP     KL0        ;  $\overline{\text{INT0}}$  中断入口转移
ORG      0013H      ;  $\overline{\text{INT1}}$  中断入口地址
LJMP     KL1        ;  $\overline{\text{INT1}}$  中断入口转移
data0    EQU    × × H    ; data0 赋值
data1    EQU    × × H    ; data1 赋值
ORG      0100H
MKL:  :
    ORL    P1, # 03H      ; L0, L1 初始化, 熄灭 L0, L1
    ANL    TCON, # 00H    ; 置  $\overline{\text{INT0}}$ 、 $\overline{\text{INT1}}$  电平触发方式

```

```

ANL      IP, #F5H      ;置INT0、INT1低优先级
MOV      IE, #85H      ;开CPU总中断,开INT0、INT1中断

```

⋮

INT0中断服务程序清单如下:

```

ORG      1000H
KL0: CLR  P1.0          ;点亮L0
MOV      R7, #data0     ;延时数据data0入R7
LCALL    DELAY          ;调延时子程序
SETB     P1.0           ;熄灭L0
RETI     ;中断返回

```

INT1中断服务程序如下:

```

ORG      1300H
KL1: CLR  P1.1          ;点亮L1
MOV      R7, #data1     ;延时数据入R7
LCALL    DELAY          ;调延时子程序
SETB     P1.1           ;熄灭L1
RETI     ;中断返回

```

延时子程序如下:

```

ORG      1200H
DELAY:   MOV  R6, #0FFH  ;延时时间常数预先设置在R7中
TM1:     MOV  R5, #0FFH
TM0:     DJNZ R5, TM0
          DJNZ R6, TM1
          DJNZ R7, DELAY
RET

```

**例 7-3** 按图 7.6.3 电路, 编写主程序循环点亮 L4~L7。根据 S0, S1 按键的状态, 点亮 L1, L2。

**解** 观察在主程序中出现中断时, 中断服务程序对主程序的影响。

程序设计包括主程序和中断服务程序设计。

(1) 循环点亮 L4~L7 及中断初始化主程序 MLED

设主程序 MLED 的入口地址为 0050H。主程序先熄灭所有 LED, 然后依次点亮 L4~L7, 不断循环, 每次点亮时间由 data 给定。主程序清单如下:

```

ORG      0000H
LJMP     MLED
ORG      0003H
LJMP     K0L0

```

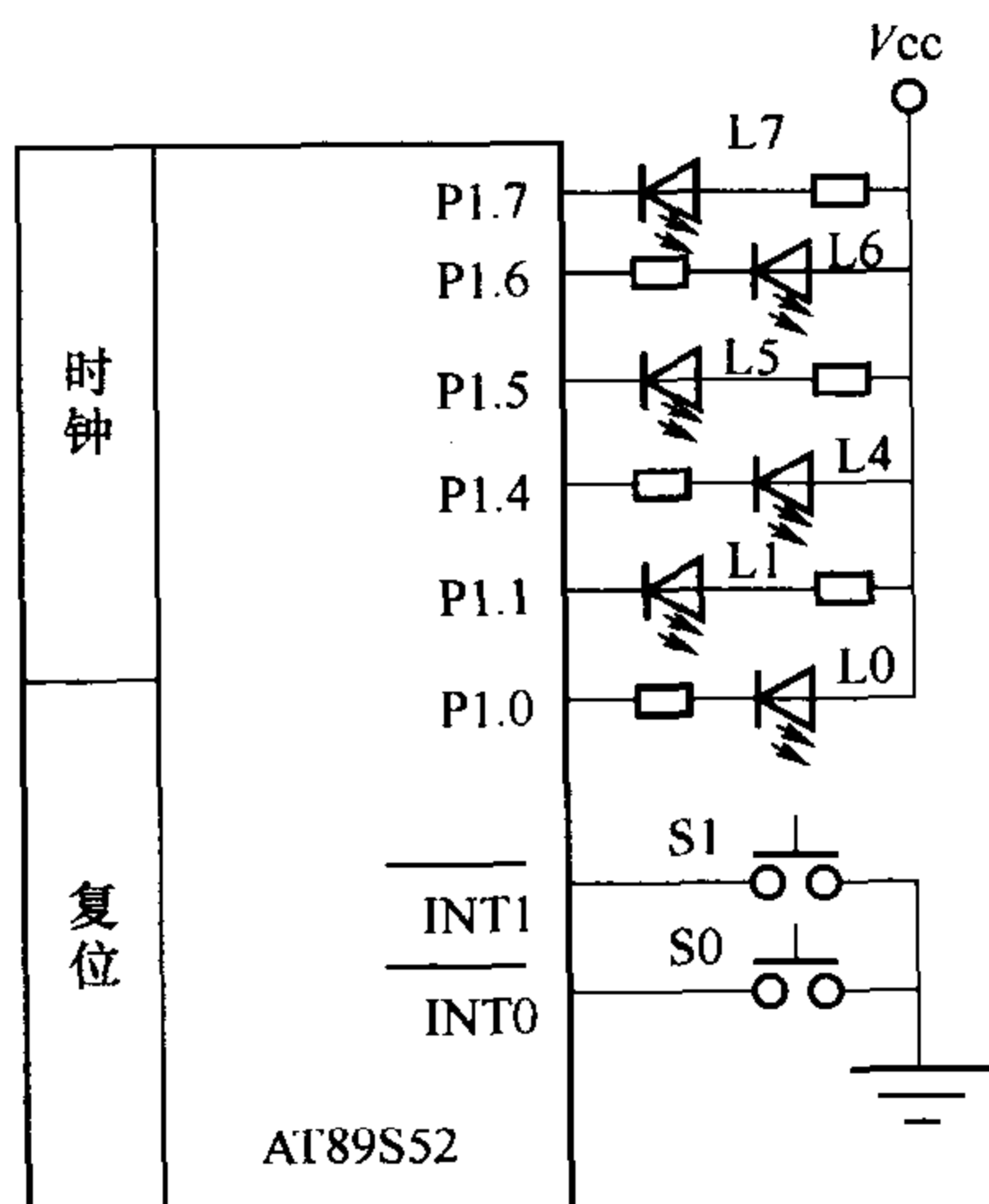


图 7.6.3 外部中断对主程序影响的演示电路

;INT0中断入口地址

```

                ORG      0013H                ;INT1中断入口地址
                LJMP     K1L1
                data0    EQU      × × H
                data1    EQU      × × H
                data     EQU      × × H        ;延时时间设定
MLED:          ORG      0050H                ;给定主程序入口地址
                ORL      P1, #0F3H          ;熄灭所有 LED
                MOV      TCON, #05H         ;INT0、INT1 下降沿触发
                MOV      IE, #85H          ;开中断
DIS:           CLR      P1.4                ;点亮 L4
                MOV      R7, #data          ;设定延时时间常数
                LCALL    DELAY              ;延时
                SETB     P1.4                ;L4 熄灭
                CLR      P1.5                ;点亮 L5
                MOV      R7, #data
                LCALL    DELAY
                SETB     P1.5                ;熄灭 L5
                CLR      P1.6                ;点亮 L6
                MOV      R7, #data
                LCALL    DELAY
                SETB     P1.6                ;熄灭 L6
                CLR      P1.7                ;点亮 L7
                MOV      R7, #data
                LCALL    DELAY
                SETB     P1.7                ;熄灭 L7
                AJMP     DIS                ;转再次循环点亮

```

## (2) 中断服务程序设计

INT0中断服务程序如下:

```

K0L0:          PUSH     PSW                ;PSW 压栈
                MOV      PSW, #18H         ;寄存器组 3 区
                CLR      P1.0
                MOV      R7, #data0
                LCALL    DELAY
                SETB     P1.0
                POP       PSW               ;弹栈 PSW
                RETI      ;中断返回

```

INT1中断服务程序如下:

```

K1L1:          PUSH     PSW                ;PSW 压栈

```

```
MOV     PSW, #10H           ;寄存器组 2 区
CLR     P1.1
MOV     R7, #data1
LCALL   DELAY
SETB    P1.1
POP     PSW                 ;弹栈 PSW
RETI                      ;中断返回
```

主程序 MLED 执行后, L4~L7 轮流点亮, 当按下 S0 或 S1 后, L4~L7 点亮循环中断, 转而点亮 L0 或 L1 后再从中断处继续 L4~L7 的轮流点亮。从这些操作可以了解中断对主程序的影响。

## 习 题

1. 中断的含义是什么? 为什么要采用中断技术?
2. 何谓查询中断? 何谓矢量中断? 何谓中断入口地址?
3. AT89S52 提供哪几种中断? 什么是中断优先级? 什么是中断嵌套? 什么是同级内的优先权管理?
4. 外部中断请求有哪两种触发方式? 对触发信号有什么要求? 如何选择和设置?
5. 何谓可屏蔽中断? AT89S52 中断系统设有几级屏蔽? 如何进行程序控制?
6. 何谓断点? 为什么要进行断点现场保护? 有哪些信息应考虑压栈保护?
7. 简述 AT89S52 应用系统中外部中断源的扩展方法。若扩展 8 个中断源, 应如何确定优先级?
8. 在 AT89S52 中, 哪些中断请求标志可以随着 CPU 响应而自动撤除? 哪些中断请求标志需由用户通过程序来撤除?
9. 请叙述中断程序设计的一般格式。在什么情况下, 中断服务中要设资源保护指令 PUSH 和 POP? 通常该指令设在何处?
10. 请写出  $\overline{\text{INT0}}$  为下降沿触发方式的中断初始化程序。
11. 在中断服务程序中能否随意设置压栈指令 PUSH? 中断服务程序中设置压栈指令后, 返回前如果没有弹栈指令会出现什么情况?
12. 当中断优先级寄存器的内容为 09H 时, 其含义是什么?

## 实 践 训 练

学完本章内容后, 可进行中断方面的实践训练。按照下述步骤进行训练。

1. 搭建下图电路, 编写程序, 当对应的 S0~S7 按键按下时, 相应的发光管 L0~L7 点亮。开关状态通过中断检测。考虑 S0~S7 按键的中断优先级应如何确定? 如何改变其优先级顺序?
2. 按例 7-2 连接好 L1, L0, S1, S2, 并连接复位电路、晶振电路、电源电路, 模拟调试



程序,并将程序下载到单片机的 Flash 中运行。并进行如下分析和实践。

- (1) 程序中 data0、data1 作什么用? 请改变它们来观察点亮保持时间,并与计算的时间对比。
  - (2) 按下 S0, S1 点亮相应的 L0, L1, 在点亮 L0 的过程中能否点亮 L1? 反之, 在点亮 L1 的过程中, 能否点亮 L0? 为什么?
  - (3) 如果要求在点亮 L0 的过程中, 按下 S1 能点亮 L1, 中断程序应怎样修改?
  - (4) 当按下 S0 或 S1 时间较长, 超过点亮延时时间会出现什么现象? 为什么?
3. 按例 7-3 连接好电路, 并连接复位电路、晶振电路、电源电路, 模拟调试程序, 将程序下载到单片机的 Flash 中运行, 观察运行情况。并进行如下分析和实践。
- (1) 给定 data0, data1, data, 调整不同的点亮延时, 观察 S0, S1 中断输入对主程序运行的影响。
  - (2) 如果要求 L4~L7 顺序点亮完第一个循环才允许点亮 L0 或 L1, 应怎样修改主程序?
  - (3) 主程序中, 点亮 L4~L7 后的延时时间相同, 为什么每次点亮后都要将 data 赋值给 R7? 如果只在 L4 点亮后赋值, 会出现什么情况?
  - (4) 如果长时间按下 S0 或 S1 键, 会出现什么情况? 请解释。
  - (5) 中断服务程序中设有资源保护的压栈指令。设主程序为复位状态下运行, 请问主程序 DELAY 和中断服务程序中的 R7 各在何处?

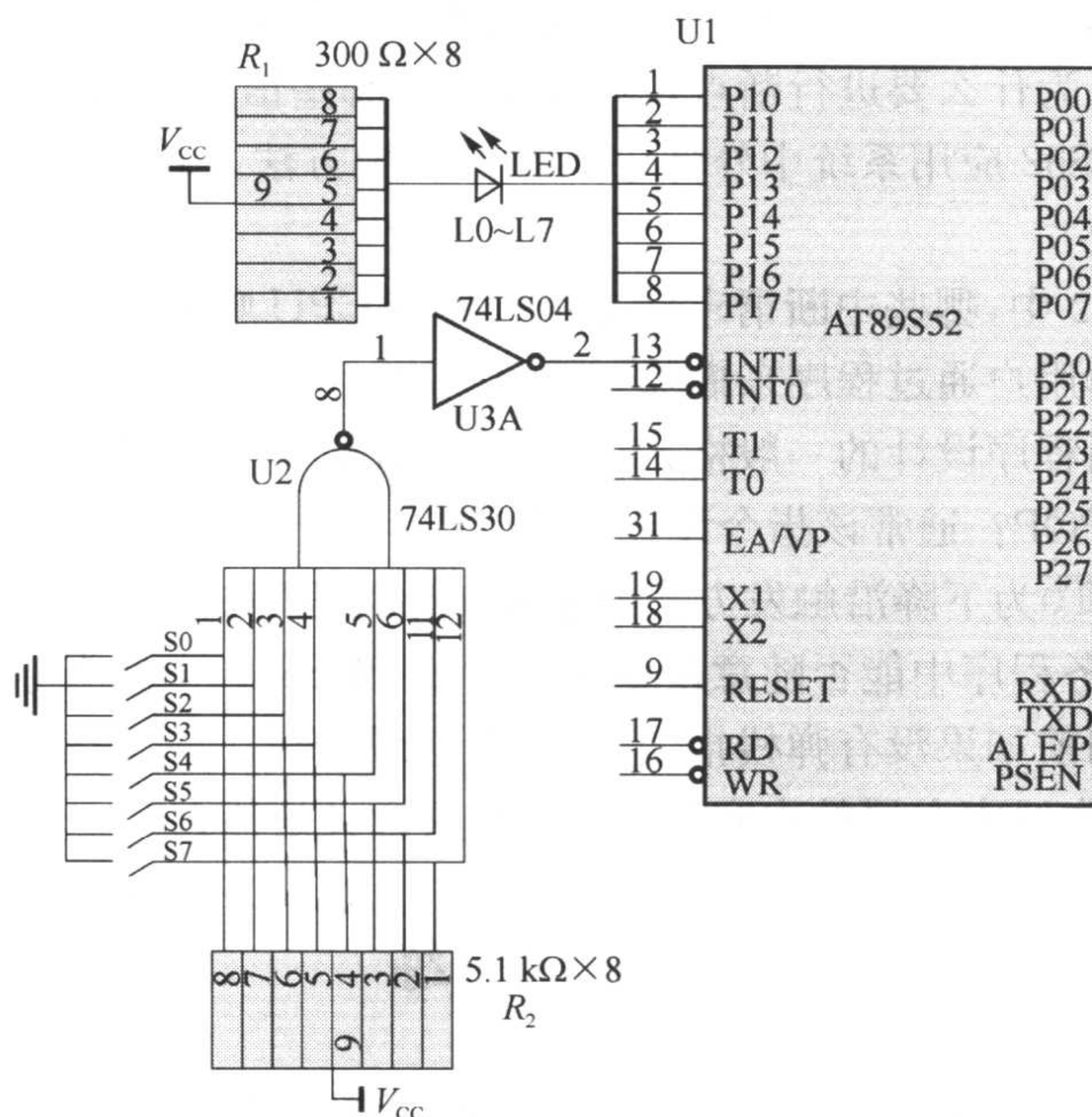


图 7.1 键盘中断测试电路

## 第 8 章 AT89S52 定时/计数器

定时/计数器是单片机的重要功能模块之一,可实现定时控制、延时、对外部事件计数和检测等功能,在工业检测、自动控制以及智能仪器等方面起着重要的作用。定时器主要完成系统运行过程中的定时功能,而计数器主要用于对外部事件的计数。本章的主要内容有:定时器的结构、定时器控制、定时器的工作模式及应用以及定时监视器。

### 8.1 定时/计数器 0/1 的结构

AT89S51 单片机有两个可编程的定时/计数器,即定时/计数器 0 和定时/计数器 1,可由程序设定作为定时器或作为计数器使用,同时还可以设定定时时间或计数值。

每个定时/计数器都具有 4 种工作模式,可通过程序选择。

AT89S52、AT89S53 及 AT89S8252 单片机有 3 个可编程定时/计数器,增加了定时/计数器 2。定时/计数器 2 有 3 种工作模式,可通过程序选择。

任意一个定时/计数器在定时时间到或者计数结束时,会产生相应的触发信号或中断请求信号。

定时/计数器 0、1 的结构框图如图 8.1.1 所示,主要由两个 16 位加 1 计数器和两个特殊功能寄存器 TMOD、TCON 组成。

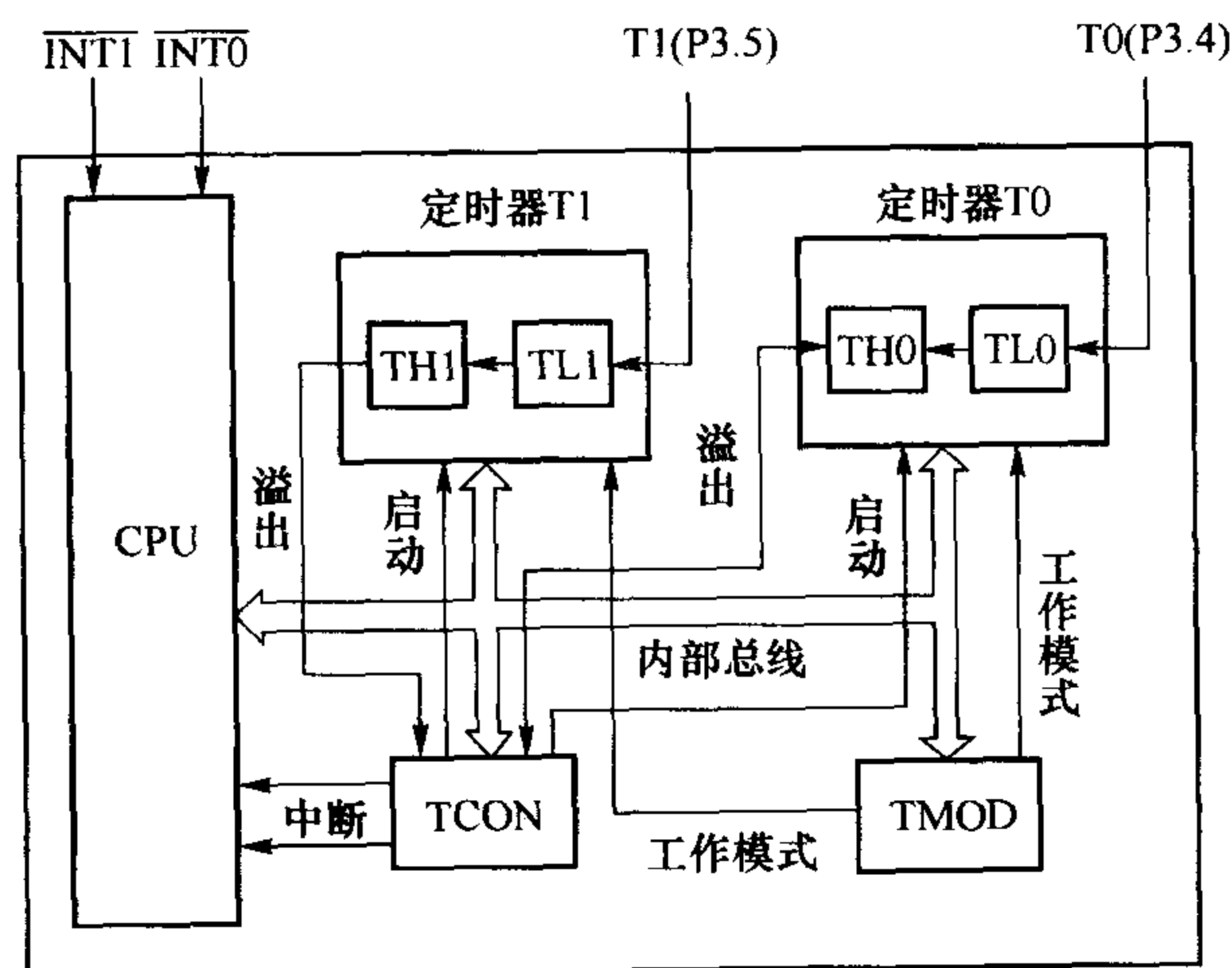


图 8.1.1 单片机 T0 和 T1 定时器结构图

定时/计数器的核心是两个 16 位的加 1 计数器。图中的特殊功能寄存器 TH0, TL0 及 TH1, TL1 用于存放两个加 1 计数器的计数结果。TH $\times$ , TL $\times$  是定时/计数器 $\times$  ( $\times=0,1$ ) 加 1 计数器计数结果的高 8 位和低 8 位。

定时/计数器设置为定时工作方式时,加 1 计数器对内部机器周期脉冲计数,即每个机器周期加 1 则计数器的数值加 1。由于机器周期是固定的,所以对机器周期的计数也就是定时,当单片机采用 12 MHz 晶振时,一个机器周期是  $1\ \mu\text{s}$ ,计数值是 100,相当于定时  $100\ \mu\text{s}$ 。

定时/计数器设置为计数工作方式时,加 1 计数器通过引脚 T0(P3.4) 和 T1(P3.5) 对外部脉冲信号计数。当外部脉冲信号产生由 1 至 0 的负跳变时,加 1 计数器的值加 1,加 1 计数溢出时可向 CPU 发出中断请求信号。具体地说,单片机每个机器周期都会对 T0 和 T1 引脚的输入电平进行采样,如果前一个机器周期采样值为 1,而下一个机器周期的采样值为 0,则加 1 计数器的值加 1。由此可见,检测一个 1 至 0 的跳变至少需要两个机器周期,所以最高计数频率为振荡频率的  $1/24$ 。对输入信号的占空比没有要求,但必须保证输入脉冲高电平和低电平的宽度都要大于 1 个机器周期。

从上面的分析可以看出,定时器本质上也是属于计数器的范畴,只不过由于它所计数的是时间长度固定的机器周期,所以就转化为定时器。

## 8.2 定时/计数器 0/1 的控制

定时器的工作模式设定和定时器的控制是由 TMOD 和 TCON 两个特殊功能寄存器来完成的,当单片机系统复位后,两个特殊功能寄存器的值都被清 0。

### 8.2.1 定时/计数器 0/1 工作模式寄存器 TMOD

TMOD 用于选择定时/计数器 0/1 的工作模式,低 4 位用于定时/计数器 0,高 4 位用于定时/计数器 1,其值可由软件设定。各位的定义格式如图 8.2.1 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
TMOD (89H)	GATE	C/ $\bar{T}$	M1	M0	GATE	C/ $\bar{T}$	M1	M0

图 8.2.1 工作模式寄存器 TMOD 的位定义

M1 和 M0:定时/计数器工作模式选择位。M1 和 M0 的不同取值设定了定时/计数器的 4 种工作模式,见表 8.2.1 所示。

C/ $\bar{T}$ :定时/计数器功能选择位。C/ $\bar{T}$ =0 为定时器方式,对单片机内部的机器周期(振荡周期的 12 倍)计数;C/ $\bar{T}$ =1 为计数器方式,计数器的输入是来自 T0(P3.4)和 T1(P3.5)引脚的外部脉冲。

GATE:门控位。GATE=0,通过软件置位 TR0(或 TR1)就可以启动定时/计数器 0(或定时/计数器 1);GATE=1,只有( $\overline{\text{INT0}}$ 或 $\overline{\text{INT1}}$ )引脚为高电平且软件使 TR0(或 TR1)置 1 时,才能启动定时器。一般情况下 GATE=0。

表 8.2.1 M1、M0 控制的 4 种工作模式

M1	M0	工作模式	功能描述
0	0	模式 0	13 位计数器
0	1	模式 1	16 位计数器
1	0	模式 2	8 位自动重装载计数器
1	1	模式 3	定时器 0:分成两个 8 位计数器 定时器 1:停止工作

应该注意的是,由于 TMOD 不能位寻址,所以只能用字节设置定时器的工作模式,而不能采用位操作指令进行置 1 与清 0。

8.2.2 定时/计数器 0/1 控制寄存器 TCON

定时/计数器控制寄存器 TCON 除可进行字节寻址外,还可以进行位寻址。各位定义及格式如图 8.2.2 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
(88H)								
位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H

图 8.2.2 控制寄存器 TCON 的位定义

TF1(TCON. 7):T1 溢出标志位。当 T1 溢出时,由硬件自动使中断触发器 TF1 置 1,在中断允许的条件下向 CPU 申请中断。当 CPU 响应中断进入中断服务程序后,TF1 又被硬件自动清 0。在中断屏蔽的情况下,TF1 一般作为软件查询标志,需软件清除。

TR1(TCON. 6):T1 运行控制位。TR1=1,定时/计数器 1 启动;TR1=0,定时/计数器 1 停止。TR1 由软件置 1 或清 0,即需要启动定时/计数器 1 时,利用指令 SETB TR1 实现,需要停止定时/计数器 1 时,CLR TR1。

TF0(TCON. 5):T0 溢出标志位。其功能和操作情况同 TF1。

TR0(TCON. 4):T0 运行控制位。其功能和操作情况同 TR1。

IE1,IT1,IE0 和 IT0(TCON. 3~TCON. 0):外部中断请求及请求方式控制位,已在 7.2.2 小节中介绍过了,在此不再赘述。

8.3 定时/计数器 0/1 的 4 种模式及应用

通过对特殊功能寄存器 TMOD 中的控制位 M1 和 M0 进行设置,可以设定定时/计数器 T0 和 T1 的 4 种工作模式,其中,在模式 0、模式 1 和模式 2 时,T0 和 T1 的工作情况完全相同;而在模式 3 时,两个定时/计数器的工作情况不同。



### 8.3.1 模式0 及应用

$M1=0, M0=0$ , 定时/计数器工作在模式0, 构成13位的定时/计数器, 结构如图8.3.1所示(如果把下标0改为1, 就是定时/计数器1在模式0时的结构图)。

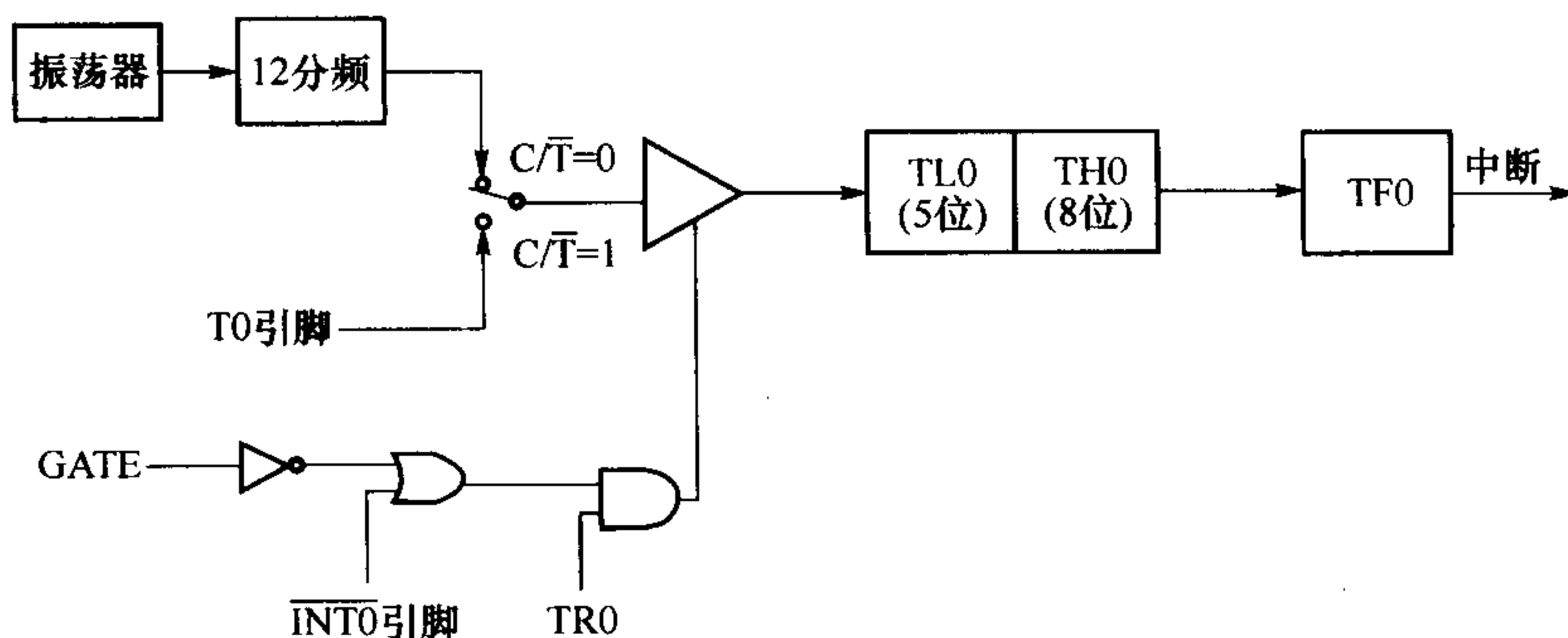


图 8.3.1 定时/计数器的模式0:13位计数器

在这种模式下, 定时/计数器0选择TH0的8位和TL0的低5位组成一个13位的定时/计数器, TL0的高3位未用。当TL0的低5位溢出时, 直接向TH0的最低位进位; TH0溢出时, 硬件置位TF0, 并向CPU申请T0中断。

通过程序可以设定TH0和TL0的初值, 初值的大小可以是 $0 \sim 8191(2^{13}-1)$ 之间的任何值。TH0和TL0从初值开始加1计数, 直至溢出为止, 所以设置的初值不同, 定时时间或计数值也不同。

需要注意的是, 当计数器溢出后, 需要通过程序重新设置TH0和TL0的初值, 否则下一次TH0和TL0将从0开始加1计数。

由于单片机采用加法计数方式, 当计数器计满后, 再来一个信号将发生溢出并产生中断, 所以实际计数的信号个数(可以是定时方式下的机器周期, 也可以是计数方式下的外部输入脉冲)就由计数器所能计数的模值和T0(或T1)的初值来决定。

$$\text{计数值} = \text{模值} - \text{初始值}$$

其中, 模值表示计数器所能计的最大值加1, 即13位计数器的模值为 $2^{13}$ , 16位计数器的模值为 $2^{16}$ 。因此, 13位计数器在初值等于8191时为最小计数值1, 在初值等于0时为最大计数值8192, 即计数范围为 $1 \sim 8192(2^{13})$ , 同时定时器的定时时间为:

$$T = (\text{模值} - \text{初值}) \times \text{机器周期}$$

在采用12 MHz晶振的情况下, 定时时间范围是 $1 \sim 8192 \mu\text{s}$ 。

**例8-1** 设定定时器T0选择工作模式0, 利用程序控制在P1.0引脚输出周期为2 ms的方波。编程实现其功能, 设单片机的振荡频率 $f_{\text{osc}} = 6 \text{ MHz}$ 。

**解** (1) 设定定时/计数器工作模式寄存器TMOD

分析: 要在P1.0引脚输出周期为2 ms的方波, 只要使P1.0引脚每隔1 ms取反一次即可。T0工作在模式0, 因此 $M1M0=00$ ; 当前的定时/计数器工作在定时方式, 因此 $C/\bar{T}=0$ ; 同时设定 $\text{GATE}=0$ , 表示计数过程不受 $\overline{\text{INT0}}$ 影响, 只由定时/计数器控制寄存器TCON

中的 TR0 位控制定时/计数器的启动与停止。由于当前的操作没有涉及到 T1, 因此 TMOD 中 T1 的相关位均设为 0, 得出 T0 的模式字 TMOD=00H。

### (2) 计算 T0 初值

定时器工作在模式 0 时为 13 位的定时/计数器。在  $f_{\text{osc}}=6\text{ MHz}$  的情况下, 每个机器周期的时间长度为:  $\frac{12}{f_{\text{osc}}}=\frac{12}{6\text{ MHz}}=2\text{ }\mu\text{s}$ 。由于定时时间为 1 ms, 所以应该计的机器周期个数即计数值为:  $\frac{1\text{ ms}}{2\text{ }\mu\text{s}}$ 。因此:

$$\text{初始值} = \text{模值} - \text{计数值} = 2^{13} - 500 = 7\,692$$

转换为二进制数为: 1111000001100B

T0 的低 5 位: 01100B=0CH

T0 的高 8 位: 11110000B=0F0H

TH0 初值为 0F0H, TL0 的初值为 0CH。

2 ms 方波的实现可以利用采用查询方式或定时器溢出中断方式。

### (3) 查询方式程序清单

```

                ORG      0000H           ;复位入口
RESET: AJMP     MAIN                    ;跳过中断服务程序区
                ORG      0100H           ;主程序
MAIN:  MOV      TMOD, #00H              ;设置 TMOD 寄存器
        MOV      TH0, #0F0H             ;送初值
        MOV      TL0, #0CH
        SETB     TR0                    ;启动定时器 0
LOOP:  JBC       TF0, NEXT              ;查询定时时间到否? 软件清除 TFO
        SJMP     LOOP
NEXT:  MOV      TH0, #0F0H              ;重装计数初值
        MOV      TL0, #0CH
        CPL      P1.0                   ;输出方波
        SJMP     LOOP                   ;重复循环

```

### (4) 定时器溢出中断方式程序清单

```

                ORG      0000H           ;复位入口
RESET: AJMP     MAIN                    ;跳过中断服务程序区
                ORG      000BH           ;T0 中断服务程序入口
        AJMP     ISOTO
                ORG      0100H           ;主程序
MAIN:  MOV      SP, #60H                ;设堆栈指针
        ACALL   INIT
        HERE:  AJMP     HERE             ;等待定时时间到, 转入中断服务程序

```

```

INIT: MOV    TMOD, #00H    ;设置 TMOD 寄存器
      MOV    TH0, #0F0H    ;送初值
      MOV    TL0, #0CH
      SETB   ET0           ;T0 开中断
      SETB   EA           ;CPU 开中断
      SETB   TR0          ;启动定时器
      RET
      ORG    0200H         ;中断服务程序
ISOTO: MOV    TH0, #0F0H    ;重新装入初值
      MOV    TL0, #0CH
      CPL    P1.0          ;输出方波
      RETI                ;中断返回

```

### 8.3.2 模式 1 及应用

M1=0, M0=1, 定时/计数器工作在模式 1, 构成 16 位的定时/计数器。其结构和操作模式几乎与模式 0 完全相同, 唯一的差别是: 在模式 1 中, 寄存器 TH0 和 TL0 以全部 16 位参与操作。

在模式 1 中, 计数器的计数范围为  $1 \sim 65\,536 (2^{16})$ , 如果  $f_{osc} = 12\text{ MHz}$ , 那么定时范围为  $1 \sim 65\,536\ \mu\text{s}$ 。

**例 8-2** 设晶振频率为  $11.059\text{ MHz}$ , 仍采用定时器控制输出方波, 要求方波的周期为  $1\text{ s}$ 。

**解** (1) 计算初值

周期为  $1\text{ s}$  的方波要求定时值为  $500\text{ ms}$ , 在时钟频率为  $11.059\text{ MHz}$  的情况下, 即使定时器工作在模式 1 (16 位计数器), 这个值也超过了模式 1 可能提供的最大定时值。如果采用降低单片机时钟频率的办法来延长定时时间, 在一定的范围内当然可以, 但这样会降低 CPU 的运行速度, 而且定时误差也会加大。下面介绍一种利用定时器定时和软件计数来延长定时时间的方法。

要获得  $500\text{ ms}$  的定时, 可选用 T0 模式 1, 定时时间为  $50\text{ ms}$ 。另设一个软件计数器, 初始值为 10。每隔  $50\text{ ms}$  定时时间到, 就产生溢出中断, 在中断服务程序中使软件计数器减 1, 这样, 当软件计数器减到 0 时, 就获得  $500\text{ ms}$  定时。

计数初值为:  $2^{16} - \frac{50 \times 10^{-3} \times 11.059 \times 10^6}{12} = 65\,536 - 46\,079 = 19\,457 = 4\text{C01H}$ , 得:

TH0=4CH, TL0=01H。

(2) 源程序如下

```

ORG 0000H
AJMP MAIN
ORG 000BH

```



```

        AJMP CTC0
        ORG 0100H
MAIN:   MOV  TMOD, #01H           ;T0 工作在模式 1, 定时方式
LOOP:   MOV  TH0, #4CH           ;装入 T0 计数初值
        MOV  TL0, #01H
        MOV  IE, #82H           ;T0 开中断
        SETB TR0                ;启动定时器
        MOV  R1, #0AH
        HERE: SJMP HERE

中断服务程序:
CTC0:   DJNZ R1, NEXT           ;R1 不等于 0, 则不对 P1.0 取反
        CPL  P1.0               ;输出方波
        MOV  R1, #0AH
NEXT:   MOV  TH0, #4CH           ;重装定时器初值
        MOV  TL0, #01H
        RETI

```

### 8.3.3 模式 2 及应用

$M1=1, M0=0$ , 定时/计数器工作在模式 2, 构成自动重新装入初值(自动重装)的 8 位定时/计数器。定时/计数器 0 工作在模式 2 时的结构如图 8.3.2 所示(如果把标记 0 改为 1, 就是定时/计数器 1 在模式 2 时的结构图)。

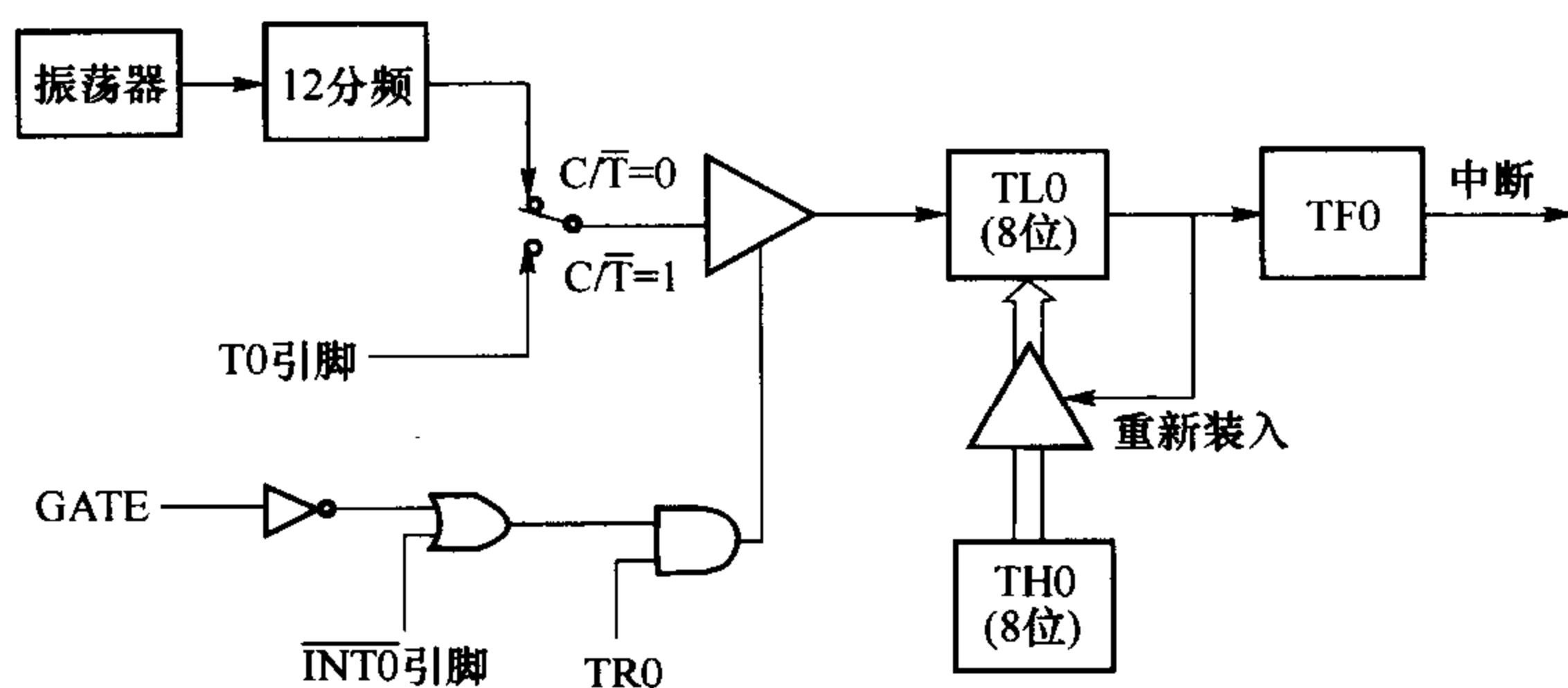


图 8.3.2 定时/计数器的模式 2: 8 位自动重载计数器

图中, TL0 作为 8 位加 1 计数器用, TH0 作为初值寄存器用。TH0 和 TL0 的初值都由软件预置。TL0 发生计数溢出时, 不仅使溢出中断标志 TF0 置 1, 而且发出重载信号, 使三态门打开, 把 TH0 中所保存的内容自动重新装入到 TL0 中, 使 TL0 从初值开始重新计数。重新装入初值后, TH0 的内容保持不变。

在模式 2 中, 计数器的计数范围为  $1 \sim 256(2^8)$ , 如果  $f_{osc} = 12 \text{ MHz}$ , 那么定时范围为  $1 \sim 256 \mu\text{s}$ 。

这种工作模式常用于定时控制。例如希望每隔  $500\ \mu\text{s}$  产生一个定时控制脉冲,若采用  $6\ \text{MHz}$  的振荡器,使  $\text{TL1}=\text{TH1}=\text{06H}$ ,  $\text{C}/\overline{\text{T}}=0$  就能实现。模式 2 还经常作为串行口波特率发生器。

**例 8-3** 设 P3.4 输入低频负脉冲信号,要求 P3.4 每次发生负跳变时, P1.0 输出一个  $500\ \mu\text{s}$  的同步脉冲。设单片机的振荡频率  $f_{\text{osc}}=6\ \text{MHz}$ 。其波形如图 8.3.3 所示。

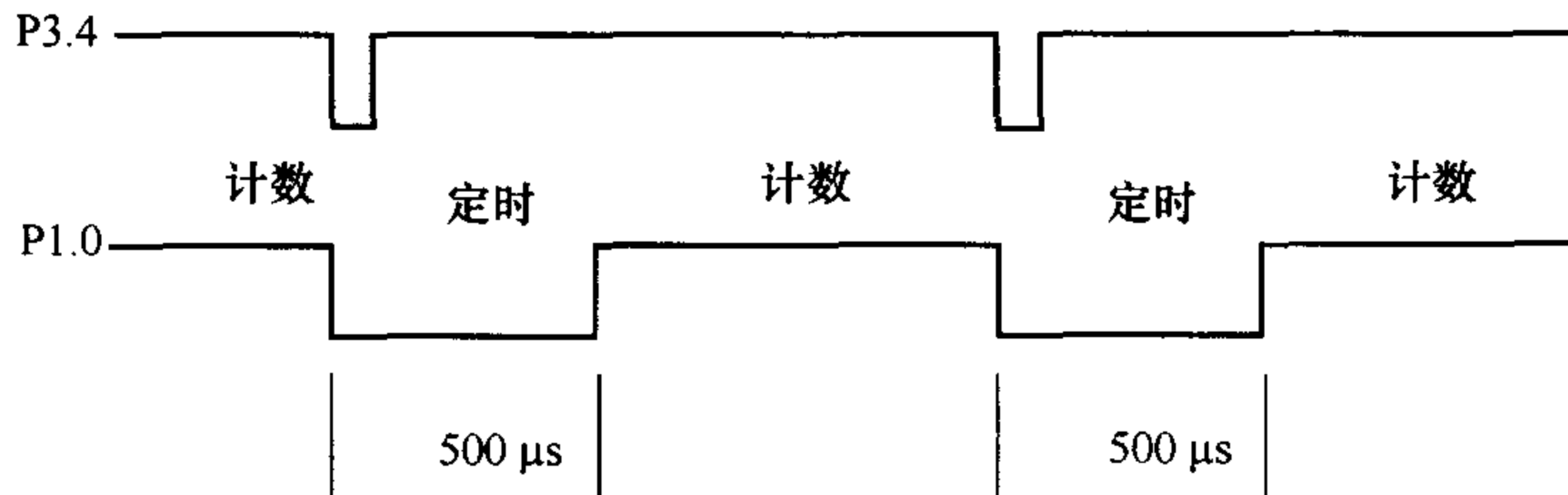


图 8.3.3 例 8.3 所实现的功能示意图

#### 解 (1) 功能分析

由图知,初始状态 P1.0 输出高电平, T0 选为方式 2, 外部事件计数初值为 FFH。当 P3.4 发生负跳变时, T0 加 1 计数溢出; 程序查询到 TF0 为 1 时, 改变 T0 为定时器工作方式 2, 定时  $500\ \mu\text{s}$ , 并使 P1.0 输出低电平。当 T0 计数溢出(定时  $500\ \mu\text{s}$  到)后, 使 P1.0 恢复高电平, 以后 T0 又恢复为外部事件计数器方式。

#### (2) 程序清单

```

START: MOV    TMOD, #06H           ;T0 为模式 2, 外部计数
        MOV    TH0, #0FFH          ;计数初值
        MOV    TL0, #0FFH
        SETB   TR0                 ;启动计数器
LOOP1:  JBC    TF0, PTF01           ;TF0 为 1, 溢出转 PTF01
        AJMP   LOOP1               ;TF0 不为 1, 等待
PTF01:  CLR    TR0
        MOV    TMOD, #02H          ;T0 设为模式 2, 定时器
        MOV    TH0, #06H           ;定时 500 μs
        MOV    TL0, #06H
        CLR    P1.0                ;P1.0 输出低电平
        SETB   TR0                 ;启动定时器
LOOP2:  JBC    TF0, PTF02           ;500 μs 到否
        AJMP   LOOP2               ;未到循环等待
PTF02:  SETB   P1.0                ;500 μs 到 P1.0 输出高电平
        CLR    TR0
        AJMP   START               ;程序循环
    
```

### 8.3.4 模式3及应用

$M1=1, M0=1$ , 定时/计数器工作在模式3。模式3只适用于T0, 当T0工作在模式3时, TH0和TL0成为两个独立的8位计数器, 使单片机增加了一个附加的8位定时/计数器。定时器T0工作于模式3时的结构如图8.3.4所示。

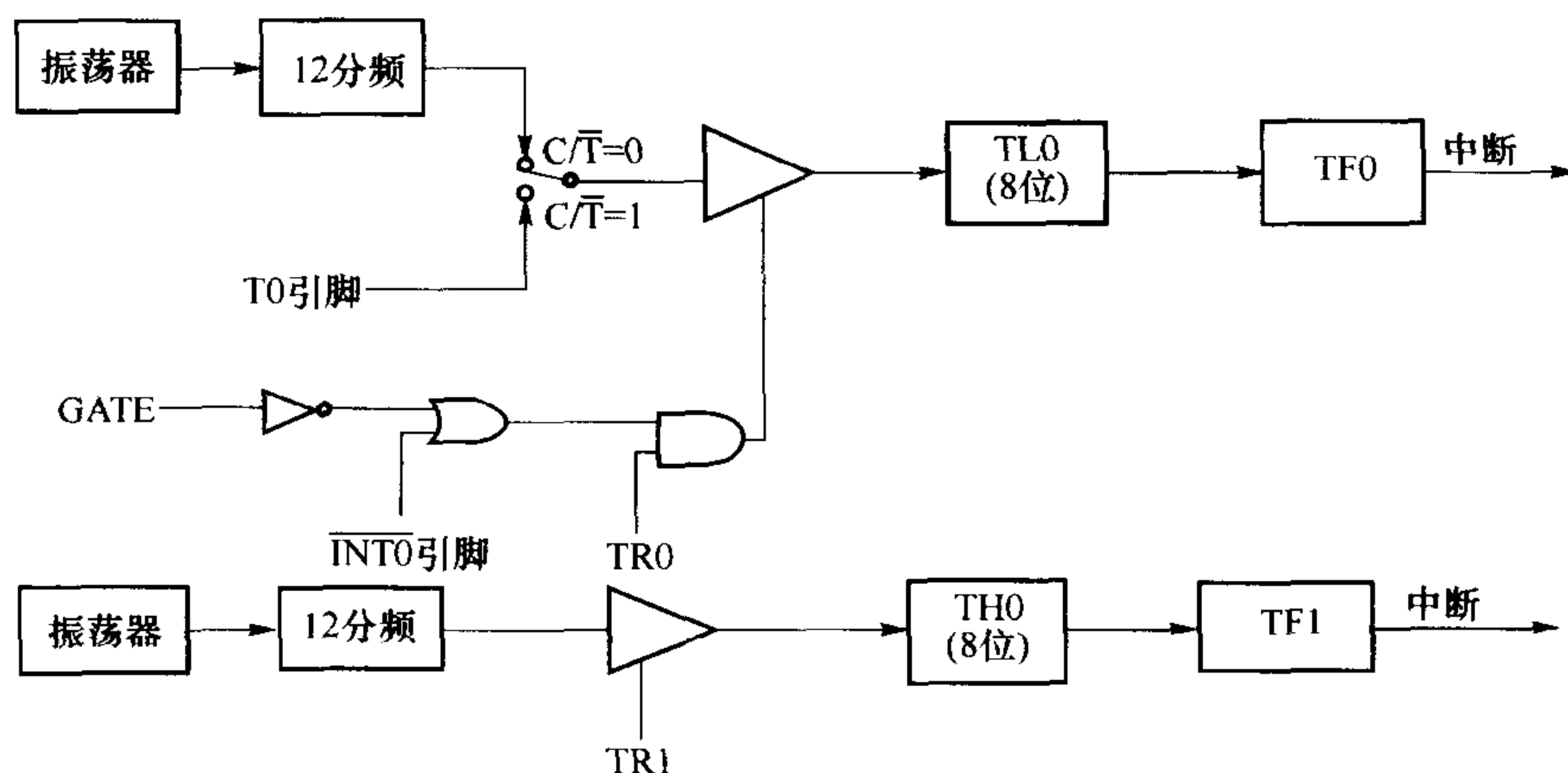


图 8.3.4 定时/计数器 0 的模式 3: 双 8 位计数器

由图可见, TL0 使用了 T0 本身的状态控制位  $C/\bar{T}$ , GATE, TF0, TR0, T0 (P3.4 引脚) 和  $\overline{INT0}$  (P3.2 引脚), 它的操作与模式 0 和模式 1 类似, 可以工作在定时器方式或计数器方式。而 TH0 只能作 8 位定时器用 (不能用作计数器方式), 并占用了 T1 的资源 TR1 和 TF1。在这种情况下, T1 可以设置为模式 0, 1, 2, 用于任何不需要中断控制的场合, 或用作串行口波特率发生器。

**例 8-4** 设某用户系统中已使用了两个外部中断源, 并置定时器 T1 工作在模式 2, 作串行口波特率发生器。现要求再增加一个外部中断源, 并由 P1.0 输出一个 5 kHz 的方波。设单片机的振荡频率  $f_{osc}=11.059\text{ MHz}$ 。

**解** (1) 功能分析

系统要求的由 P1.0 输出一个 5 kHz 方波的功能可以由定时器实现, 而需要增加的一个外部中断源可以通过计数器来模拟实现, 所以可以设置 T0 工作于模式 3, 分成两个 8 位计数器。TH0 工作于定时器方式, 由它来实现由 P1.0 输出一个 5 kHz 方波的功能; 而 TL0 工作于计数方式, 把 T0 的引脚作为附加的外部中断输入端, TL0 的计数初值为 0FFH, 当检测到 T0 引脚电平出现由 1 至 0 的负跳变时, TL0 产生溢出, 申请 T0 中断。这相当于边沿触发的外部中断源。

TL0 的计数初值为: 0FFH。

TH0 的计数初值计算如下。

方波的频率为 5 kHz, 则周期为 0.2 ms, 其半周期为  $0.1\text{ ms}=100\text{ }\mu\text{s}$ , 因此计数初值为:

$$256 - \frac{100 \times 10^{-6} \times 11.059 \times 10^6}{12} = 256 - 92 = 164 = 0A4H。$$

(2) 程序如下

```
START: MOV  TMOD, #27H           ;T0 为模式 3 计数, T1 为模式 2 定时
        MOV  TL0, #0FFH
        MOV  TH0, #0A4H
        MOV  TL1, #band          ;band 是根据波特率要求设置的常数
        MOV  TH1, #band
        MOV  TCON, #55H          ;设定中断触发方式并启动 T0, T1
        MOV  IE, #9FH            ;开中断
        :
```

TL0 溢出中断服务程序(由 000BH 转来):

```
TL0INT: MOV  TL0, #0FFH          ;TL0 重新赋初值
        :
        RETI                      ;中断处理
```

TH0 溢出中断服务程序(由 001B 转来):

```
TH0INT: MOV  TH0, #0A4H          ;TH0 重新赋初值
        CPL  P1.0                 ;输出方波
        RETI
```

串行口和外部中断 0、外部中断 1 的服务程序在此不再一一列出。

### 8.3.5 定时/计数器的其他应用

**例 8-5** 利用定时/计数器 T1 测量  $\overline{\text{INT1}}$  引脚上出现的正脉冲宽度, 并以机器周期数来表示。

**解** (1) 功能分析

当门控位  $\text{GATE}=1$  时, 定时器的启动计数受外部输入  $\overline{\text{INT}} \times$  引脚电平控制。在  $\text{TR} \times=1$  时, 若  $\overline{\text{INT}} \times=1$ , 则启动计数, 若  $\overline{\text{INT}} \times=0$ , 则停止计数。利用这一特点可测试外部输入脉冲的宽度。

设被测脉冲由 P3.3 输入, T1 为定时器方式 1。测试时, 检测  $\overline{\text{INT1}}$  由低变高时, 使  $\text{TR1}=1$ , 启动计数,  $\overline{\text{INT1}}$  再次变低时, 停止计数, 此计数值即为以机器周期表示的被测正脉冲宽度。当晶振为 12 MHz 时, 实际的正脉冲宽度即为  $(\text{TH0} \times 256 + \text{TL0}) \times 1 \mu\text{s}$ 。

(2) 程序清单

```
START: MOV  TMOD, #90H           ;门控定时 T1 方式 1
        MOV  TL1, #00H
        MOV  TH1, #00H
DONE:  JB   P3.3, DONE            ;等  $\overline{\text{INT1}}$  变低
```

```

DONE1: JNB    P3.3, DONE1      ; 等正脉冲到来
      SETB    TR1              ; 启动 T1 计数
DONE2: JB     P3.3, DONE2      ; 等待  $\overline{\text{INT1}}$  再次变低
      CLR     TR1              ; 停止 T1 计数
      MOV     R0, #40H         ; 计数值保存在 40H, 41H 单元
      MOV     A, TL1
      MOV     @R0, A
      INC     R0
      MOV     A, TH1
      MOV     @R0, A

```

## 8.4 定时/计数器 T2

在 AT89S52, AT89S53 及 AT89S8252 等单片机中, 增加了一个 16 位定时/计数器 T2, 它的功能比 T0, T1 更强。与 T2 有关的特殊功能寄存器有: 计数寄存器 TH2 和 TL2, T2 控制寄存器 T2CON, T2 模式寄存器 T2MOD, 捕捉/重装载寄存器 RCAP2L 和 RCAP2H。

### 8.4.1 T2 控制寄存器

T2 的工作是通过软件对 T2CON 寄存器进行设置来控制的, T2 控制寄存器 T2CON 的格式及功能如图 8.4.1 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
T2CON (C8H)	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/ $\overline{\text{T2}}$	CP/ $\overline{\text{RL2}}$
位地址	CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H

图 8.4.1 T2 控制寄存器 T2CON 的位定义

**TF2:** 定时/计数器 T2 溢出标志位。T2 溢出时置位, 并申请中断。此标志必须用软件清除。当 T2 工作于波特率发生器模式时, 亦即 RCLK=1 或 TCLK=1 时, T2 溢出时并不置位 TF2。

**EXF2:** 定时/计数器 T2 外部标志位。当 EXEN2=1, 且 T2EX 引脚上出现负跳变而造成捕获或重装载时, EXF2 置位, 申请中断, 此标志由软件清除。

**RCLK:** 串行口接收时钟选择位。由软件置位或清除, 用它可选择 T2 或 T1 作为串行口接收波特率发生器。RCLK=1 时, 用 T2 溢出脉冲作为串行口的接收时钟; RCLK=0 时, 用 T1 的溢出脉冲作为串行口的接收时钟。

**TCLK:** 串行口发送时钟选择位。由软件置位或清除, 用它可选择 T2 或 T1 作为串行口发送波特率发生器。TCLK=1 时, 用 T2 溢出脉冲作为串行口的发送时钟; TCLK=

0 时,用 T1 的溢出脉冲作为串行口的发送时钟。

EXEN2:定时/计数器 T2 外部允许控制位。由软件置位或清除,以允许或不允许用外部信号来触发捕获或重装载操作。当 EXEN2=1 时,若 T2 未用于串行口的波特率发生器,则在 T2EX 引脚出现信号负跳变时,将造成 T2 捕获或重装载,并置位 EXF2 标志,请求中断;当 EXEN2=0 时,T2EX 引脚上的信号无效。

TR2:定时/计数器 T2 启动/停止控制位。TR2=1,启动计数器;TR2=0,停止计数器。

C/T2:定时/计数器 T2 定时器方式与计数器方式选择位。当 C/T2=0 时,选择定时器工作方式;当 C/T2=1 时,选择计数器工作方式。

CP/RL2:定时/计数器 T2 捕获/重装载控制位。当 CP/RL2=1 时,选择捕获功能,此时若 EXEN2=1,在 T2EX 引脚上出现信号负跳变时,将发生捕获操作,即把 TH2 和 TL2 的内容传送给 RCAP2H 和 RCAP2L;当 CP/RL2=0 时,选择重装载功能,这时若 T2 溢出或在 EXEN2=1 条件下 T2EX 引脚出现信号负跳变,都会发生自动重装载操作,即把 RCAP2H 和 RCAP2L 的内容传送给 TH2 和 TL2。

### 8.4.2 T2 模式寄存器

定时/计数器 T2 模式寄存器 T2MOD 的格式及位定义,如图 8.4.2 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
T2MOD (C9H)	—	—	—	—	—	—	T2OE	DCEN

图 8.4.2 T2 模式寄存器 T2MOD 的位定义

—:未定义位。

T2OE:定时/计数器 T2 输出允许控制位。当 T2OE=1 时,启动定时/计数器 T2 的可编程时钟输出功能,允许时钟输出至引脚 T2/P1.0;当 T2OE=0 时,禁止引脚 T2/P1.0 输出。

DCEN:定时/计数器 T2 加减计数控制位。当 DCEN=1 时,允许 T2 作为加/减计数器使用。具体的计数方向由 T2EX 引脚来控制,当 T2EX=1 时,T2 进行加计数;当 T2EX=0 时,T2 进行减计数。DCEN=0 时,T2 自动向上计数。

### 8.4.3 T2 的工作模式

定时/计数器 T2 与 T0、T1 具有类似的功能,既可以当定时器使用,也可以作为外部事件计数器使用,其工作方式由特殊功能寄存器 T2CON 中的 C/T2 位来选择。AT89S52 的定时/计数器 T2 有 4 种工作模式:自动重装载、捕获、波特率发生器、可编程时钟输出模式。工作模式的选择由 T2CON 中的 RCLK, TCLK, CP/RL2, T2OE, C/T2 和 TR2 位来决定,如表 8.4.1 所示。

表 8.4.1 定时/计数器 T2 工作模式

$C/\overline{T2}$	RCLK+TCLK	$CP/\overline{RL2}$	T2OE	TR2	工作模式
×	0	0	0	1	16 位自动重装载模式
×	0	1	0	1	16 位捕获模式
×	1	×	×	1	波特率发生器
×	×	×	×	0	停止计数
0	1	×	1	1	时钟输出模式

### 1. 16 位捕获模式

若 RCLK 和 TCLK 都没有置位,且  $CP/\overline{RL2}=1$ ,定时/计数器 T2 工作于 16 位捕获模式,其结构原理图如图 8.4.3 所示。

在捕获方式下,通过 T2CON 控制位 EXEN2 来选择两种不同的方式。如果 EXEN2=0,T2 作为普通的 16 位定时/计数器使用,并由  $C/\overline{T2}$  位来决定它是用于定时器还是用于计数器;如果作为定时器使用,其计数输入为振荡器频率的 12 分频信号;如果作为计数器使用,是对 T2 引脚(与 P1.0 复用)上的输入脉冲计数。计数溢出时,置位 T2CON 中的 TF2 位,同时向 CPU 发出中断请求信号。

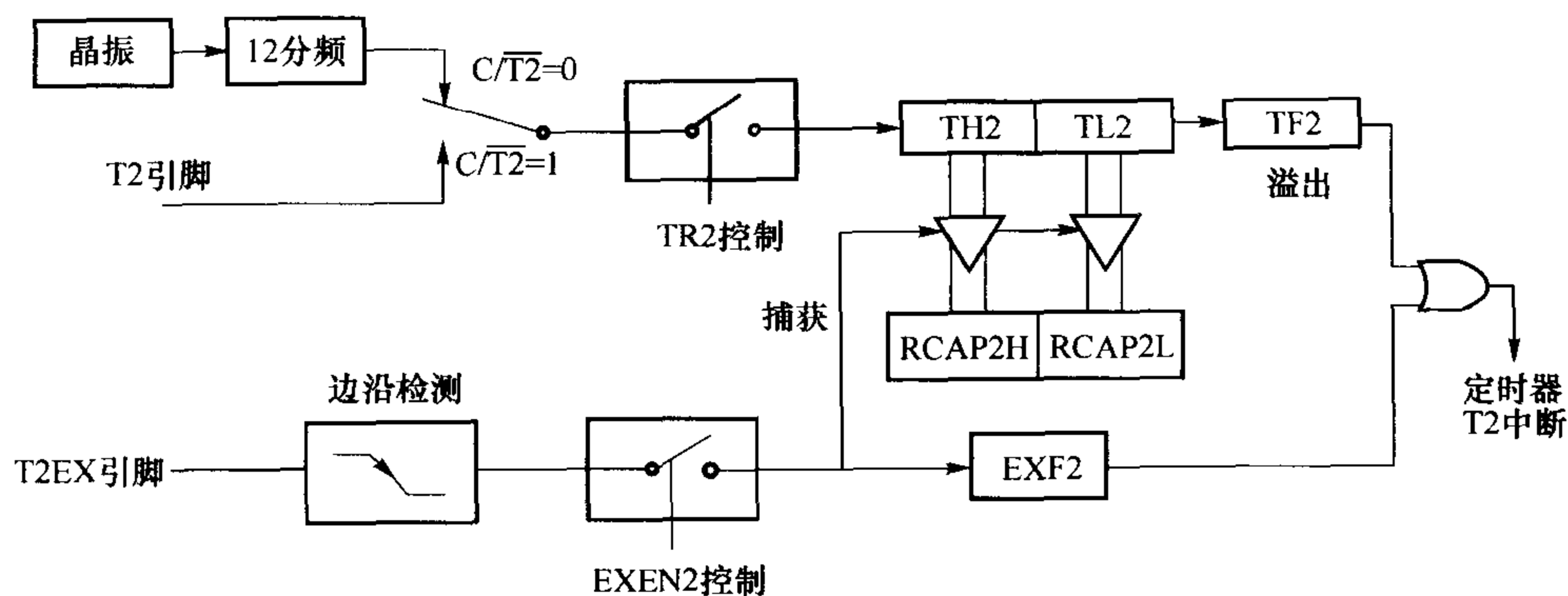


图 8.4.3 定时/计数器 T2 的捕获模式结构原理图

如果 EXEN2=1,T2 在实现前述定时和计数功能外增加了捕获功能,即当 T2EX 引脚(与 P1.1 复用)的输入信号发生负跳变时,就会把 TH2 和 TL2 的当前内容锁存到捕获寄存器 RCAP2H 和 RCAP2L 中。另外,T2EX 引脚信号的跳变使得 T2CON 中的 EXF2 置位,与 TF2 类似,EXF2 也会向 CPU 发出中断请求信号。

### 2. 16 位自动重装载模式

若 RCLK 和 TCLK 都没有置位,且  $CP/\overline{RL2}=0$ ,定时/计数器 T2 工作于 16 位自动重装载模式,其结构原理图如图 8.4.4 所示。

当定时/计数器 T2 工作于 16 位自动重装载模式时,能对其编程设定为加计数或减计数,这个功能可通过定时/计数器 T2 模式寄存器 T2MOD 中的 DCEN 来选择。复位后 DCEN 为 0,关闭了加减计数选择的功能,T2 工作在默认的加计数方式。在这种方式下,若 EXEN2=0,T2 加计数直至 0FFFFH 溢出,置位 TF2,向 CPU 发出中断请求信号,同



时把寄存器 RCAP2H 和 RCAP2L 的值重装载到 TH2 和 TL2 中,RCAP2H 和 RCAP2L 的值可由软件预置;若 EXEN2=1,重装载信号可以由 T2 溢出触发,也可以由外部输入端 T2EX 引脚上从 1 至 0 的负跳变触发,此负跳变同时使 EXF2 置位,同样向 CPU 发出中断请求信号。

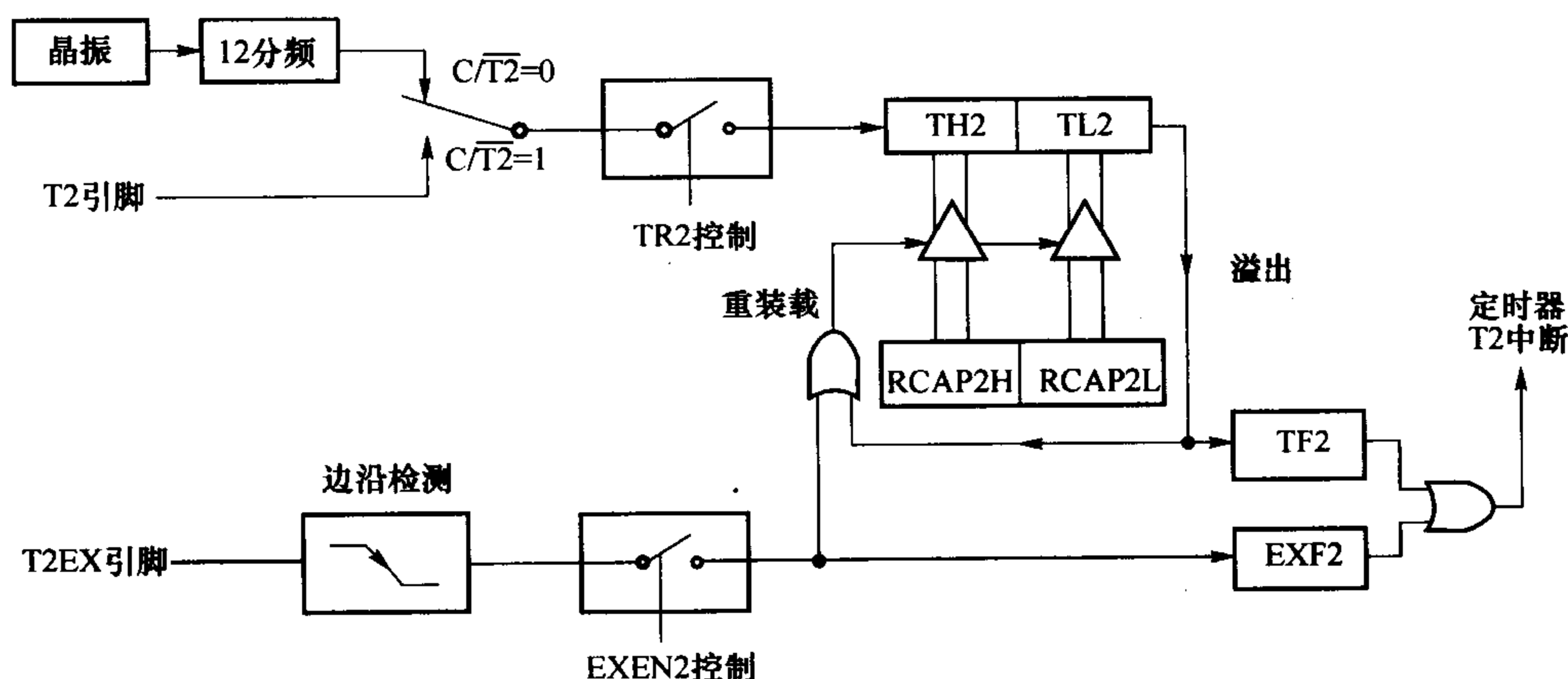


图 8.4.4 定时/计数器 T2 的自动重装载模式结构原理图(DCEN=0)

如果 DCEN=1,允许 T2 进行加减计数的选择,如图 8.4.5 所示。在这种方式下, T2EX 引脚控制计数方向。当 T2EX 引脚为逻辑 1 时, T2 进行加计数直至 0FFFFH,然后溢出,置位 TF2,同时把寄存器 RCAP2H 和 RCAP2L 的值重装载到 TH2 和 TL2 中;当 T2EX 引脚为逻辑 0 时, T2 进行减计数,若寄存器 TH2 和 TL2 中的数值等于 RCAP2H 和 RCAP2L 中的值时,计数溢出,置位 TF2,同时将 0FFFFH 重新装载到 TH2 和 TL2 中。

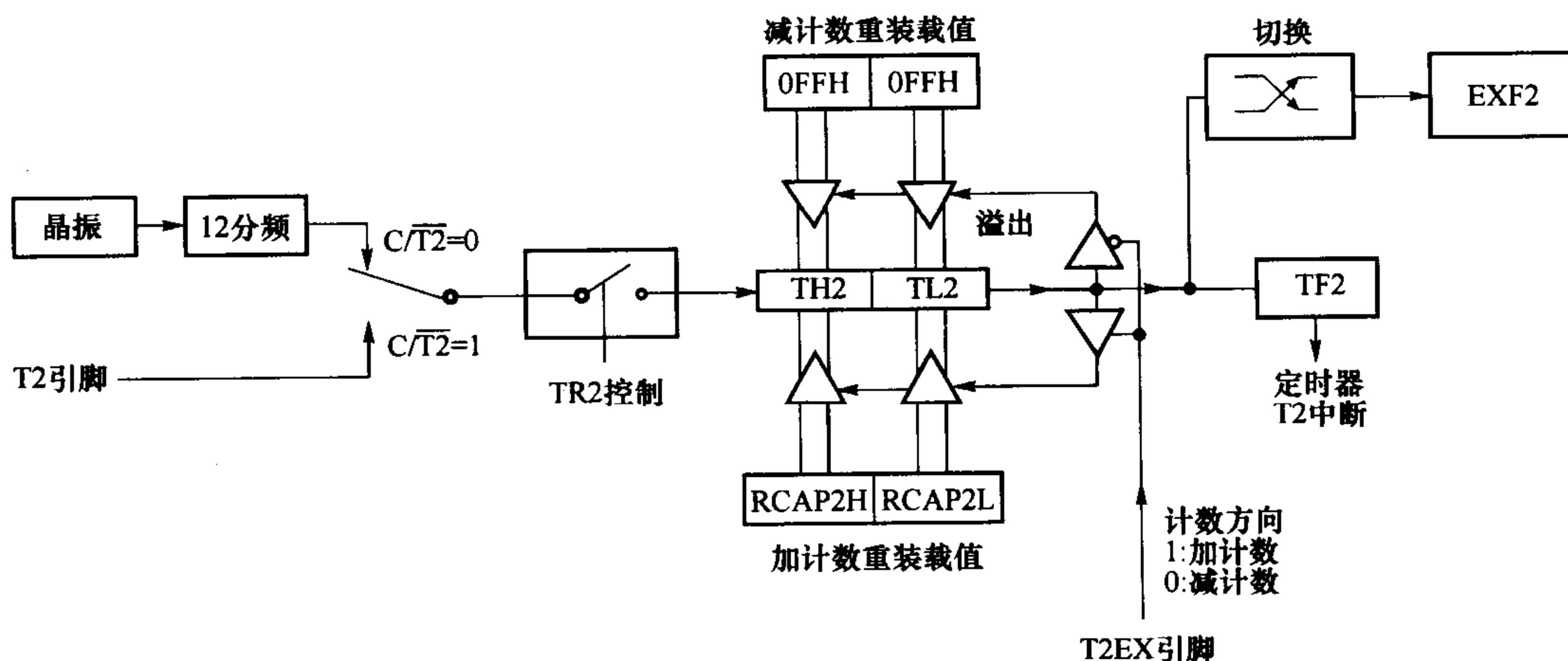


图 8.4.5 定时/计数器 T2 的自动重装载模式结构原理图(DCEN=1)

无论 T2 发生上溢出还是下溢出,EXF2 标志位的内容都要被切换,所以此时 EXF2 不再是中断标志,在此种模式下,它可以用来作为增加计数器分辨率的第 17 个计数位使用。

### 3. 波特率发生器模式

当 T2CON 中的 TCLK 和 RCLK 中的某一位置位,或者两位同时置位时,定时/计数器 T2 工作于波特率发生器模式,其结构原理图如图 8.4.6 所示。

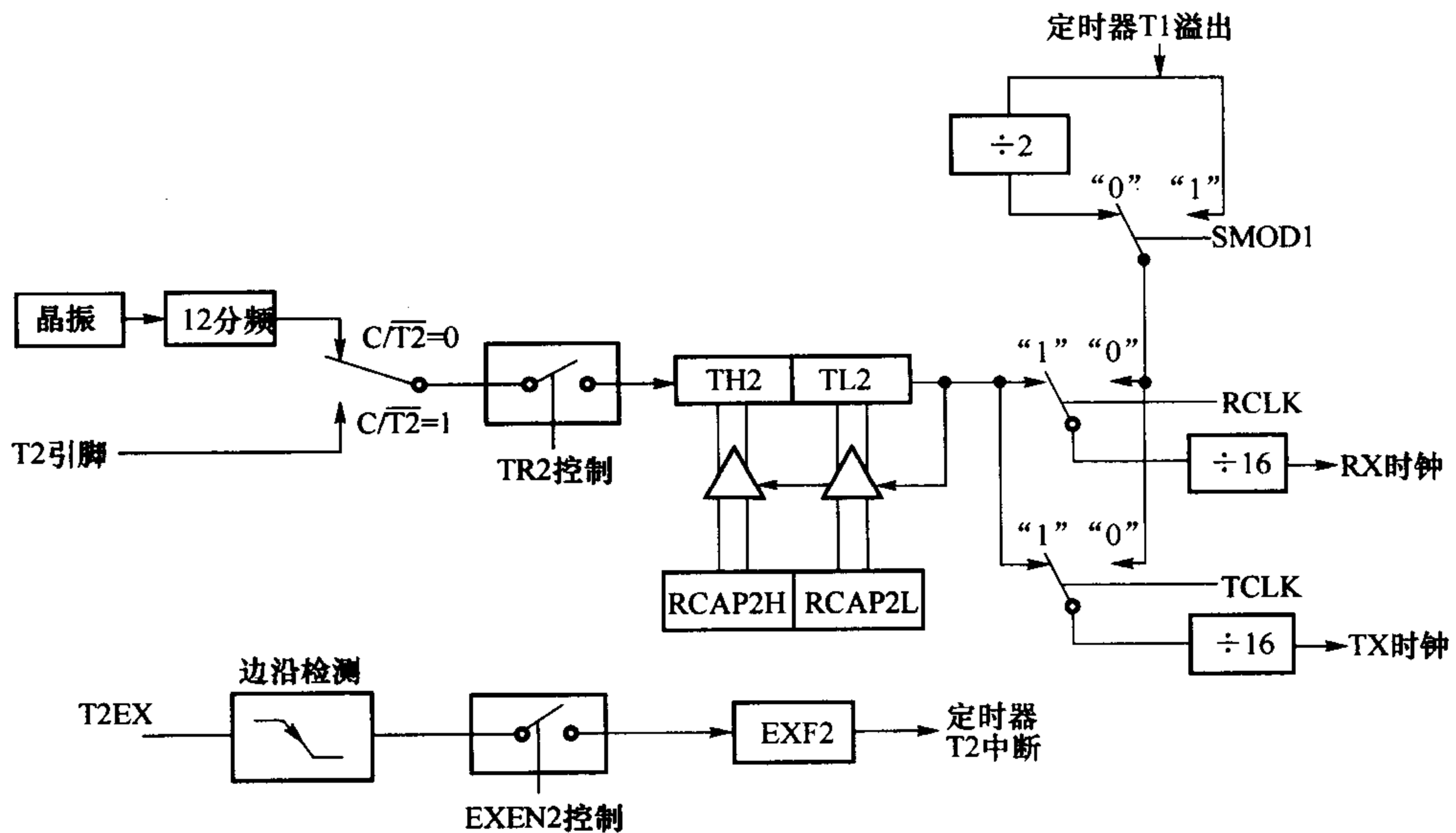


图 8.4.6 定时/计数器 T2 的波特率发生器模式结构原理图

由图可见,RCLK 和 TCLK 用来控制两个模拟开关的位置,如果其值为 0,选用 T1 作为串行口波特率发生器,如果其值为 1,则选用 T2 作为串行口波特率发生器。发送和接收的波特率可以不同,进一步的讨论参见 9.3.4 小节。

### 4. 可编程时钟输出

以上的 3 种工作模式是 8052 单片机的 T2 所具有的工作模式,AT89S52 在保留上述 3 种工作模式的基础上,还增加了一种可编程时钟输出模式,此时能够从 P1.0 引脚输出占空比为 50% 的时钟脉冲,相当于一个时钟发生器,其结构原理图如图 8.4.7 所示。

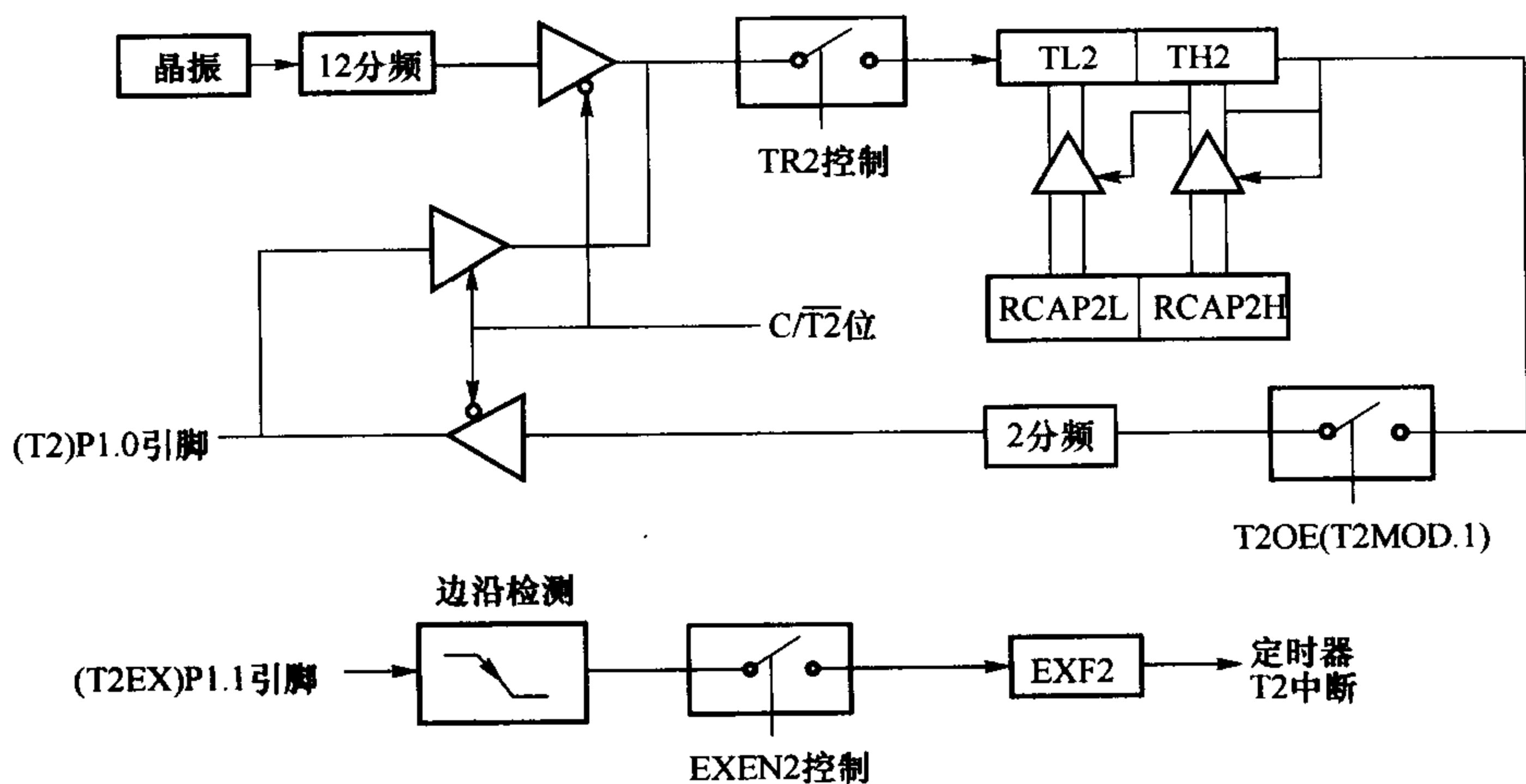


图 8.4.7 定时/计数器 T2 的可编程时钟输出模式结构原理图

T2 工作于可编程时钟输出方式时要求  $C/\overline{T2}=0$ , 即定时/计数器 T2 工作于定时器方式下, 且  $T2OE(T2MOD.1)=1$ , 即允许时钟输出。用 TR2 控制 T2 的启动与停止, 从而达到对输出时钟控制的目的。

由图可见, P1.0 除了可以作为一般 I/O 接口使用外, 还有两种功能: 其一可以作为定时/计数器 T2 的外部时钟输入口; 其二可以用于占空比为 50% 的时钟脉冲输出口。当系统的时钟频率为 16 MHz 时, 输出时钟频率范围为 61 Hz~4 MHz。

时钟输出频率取决于振荡器频率和 T2 捕获/重装载寄存器 RCAP2H、RCAP2L 的值, 计算公式如下:

$$\text{时钟输出频率} = \frac{f_{osc}}{4 \times [65536 - (RCAP2H, RCAP2L)]}$$

在可编程时钟输出模式下, T2 的翻转不会产生中断, 这个特性和 T2 作为波特率发生器时类似。所以 T2 在作为波特率发生器的同时, 也可作为时钟发生器, 但是这时所产生的波特率和输出的时钟频率不是独立的。

## 8.5 定时监视器

单片机应用系统一般应用于工业现场, 虽然单片机本身具有很强的抗干扰能力, 但仍然存在系统由于受到外界干扰使所运行的程序失控引起程序“跑飞”的可能性, 从而使程序陷入“死循环”, 这时系统将完全瘫痪。如果操作者在场, 可以通过人工复位的方式强制系统复位, 但操作者不可能一直监视着系统, 即使监视着系统, 也往往是在引起不良后果之后才进行人工复位。为此常采用程序监视技术, 就是俗称的“看门狗”(Watch Dog)技术。测控系统的应用程序往往采用循环方式运行, 每一次循环运行的时间基本固定。“看门狗技术”就是不断监视程序运行的循环时间, 如果出现运行时间超过设定的循环时间, 则产生复位信号, 强制系统复位。这好比是主人养了一条狗, 主人在正常工作的时候总是不忘每隔一段固定时间就给狗吃点东西, 狗吃过东西就安静下来, 不影响主人工作。如果主人打瞌睡, 到一定时间, 狗饿了, 就会大叫起来, 把主人吵醒。可见, “看门狗”电路一般具有如下特性:

- (1) 本身能独立工作, 基本上不依赖于 CPU;
- (2) CPU 在一个固定的时间间隔内和该系统打一次交道(喂狗), 表明系统正常;
- (3) 当 CPU 陷入死循环, 能及时发现并使系统复位。

### 8.5.1 AT89S52 的定时监视器

AT89S52 的定时监视器(WDT)是由一个 13 位的计数器和定时监视器复位特殊功能寄存器 WDTRST 组成, WDTRST 的地址为 0A6H。系统复位后定时监视器的默认状态为无效状态, 用户必须依次将 1EH 和 0E1H 写入特殊功能寄存器 WDTRST, 才能启动定时监视器。启动后每个机器周期对定时监视器中的 13 位的计数器进行加 1 计数, 当计数器发生溢出时将在 RST 引脚上输出高电平, 从而使系统复位。只有硬件复位(Reset)或 WDT 溢出复位才能使已启动的 WDT 无效。

依次将 1EH 和 0E1H 写入特殊功能寄存器 WDTRST 可以启动定时监视器,为了避免在系统正确运行过程中 WDT 溢出而使系统复位,必须在 WDT 溢出之前再将 1EH 和 0E1H 依次写入特殊功能寄存器 WDTRST(喂狗),从而使 13 位的计数器重新开始计数。也就是说,在振荡电路已经正常起振并且启动了 WDT 的情况下,每次计数在达到 8 191 (1FFFH)个机器周期以前,用户必须进行“喂狗”操作。定时监视器复位 WDTRST 只能写不能读,而 WDT 中的 13 位计数器则是既不能读也不能写的计数器。13 位计数器计满回 0 溢出将在 RST 引脚上产生复位信号,这个复位高电平脉冲宽度为 98 个振荡周期。

在低功耗状态下 WDT 和振荡电路均停止工作,这时用户不需要维护 WDT,可以通过硬件复位或优先进入低功耗状态的外部中断来终止低功耗状态。通常情况下若通过硬件复位来终止低功耗状态,则任何时候维护 WDT 都会使单片机复位。为保证在终止低功耗的过程中,包括退出低功耗状态在内 WDT 不产生溢出,最好在刚刚进入退出低功耗模式前复位 WDT。

在进入休眠状态前,特殊功能寄存器 AUXR 中的 WDIDLE 位将决定在休眠过程中 WDT 是否继续运行和计数。

### 8.5.2 辅助功能寄存器 AUXR

辅助功能寄存器 AUXR 是一个多功能选择控制寄存器,地址是 8EH,不能位寻址。各位定义及格式如图 8.5.1 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
AUXR (8EH)	—	—	—	WDIDLE	DISRT0	—	—	DISALE

图 8.5.1 辅助功能寄存器 AUXR 的位定义

—:未定义位。

WDIDLE:休眠模式下 WDT 控制位。当 WDIDLE = 0 时,在休眠模式下 WDT 继续运行计数;当 WDIDLE = 1 时,在休眠模式下 WDT 停止运行计数。

DISRT0:RST 输出控制位。当 DISRT0 = 0 时,定时监视器定时输出后 RST 置成高电平;当 DISRT0 = 1 时,仅仅为 RST 引脚输入。

DISALE:ALE 输出控制位。当 DISALE = 0 时,ALE 输出  $f_{osc}/6$  的波形信号,占空比为 1:2;当 DISALE = 1 时,ALE 只有在 MOVX 和 MOVC 指令下有效。

以下给出定时监视器程序,主要包括初始化程序和喂狗程序两个部分:

```

MAIN: MOV    AUXR, #10H        ;初始化 AUXR
      MOV    WDTRST, #1EH      ;启动定时监视器
      MOV    WDTRST, #0E1H
      ⋮
      LCALL  DOG                ;调用 DOG 程序的时间间隔应小于整个
                                ;程序的运行时间
      END

```

```
DOG:    MOV    WDTRST, #1EH    ;喂狗程序
        MOV    WDTRST, #0E1H
        RET
```

## 习 题

1. AT89S52 单片机内部设有几个定时/计数器? 它们是由哪些特殊功能寄存器组成的?
2. 定时/计数器 T0 和 T1 有几种工作模式? 各完成什么功能?
3. 定时/计数器 T2 有几种工作模式? 各有什么特点?
4. 定时/计数器用作定时器时,其定时时间与哪些因素有关? 作为计数器时,对外界计数频率有何要求?
5. 利用 T0 方式 0 产生 1 ms 的定时,在 P1.0 引脚上输出周期为 2 ms 的方波。设单片机振荡频率为 11.059 MHz,请编程实现。若方波周期为 1 s,该如何实现?
6. 单片机用内部定时方法产生频率为 100 kHz 的方波,设其振荡频率为 12 MHz,请编程实现。
7. 以定时/计数器 T1 进行外部事件计数。每计数 1 000 个脉冲后,定时/计数器 T1 转为定时工作方式,定时 10 ms 后,再转为计数方式,如此循环不止。设单片机振荡频率为 11.059 MHz,请编程实现。
8. 利用 T0 和 P1.0 输出矩形波,高电平宽度为 50  $\mu$ s,低电平宽度为 100  $\mu$ s,振荡频率为 6 MHz。
9. 已知单片机的振荡频率为 6 MHz,试编写程序,利用定时器 T0 工作在方式 3,使 P1.0 和 P1.1 分别输出周期为 1 ms 和 400  $\mu$ s 的方波。
10. 已知单片机的振荡频率为 6 MHz,试编写程序,利用定时器 T2 产生 500 ms 的延时,在 P1.1 引脚产生周期为 1 s 的方波。

## 实 践 训 练

学完本章内容后,可进行实践训练,理解 AT89S52 单片机内部定时/计数器的工作原理、工作方式,掌握定时/计数器的定时、计数方法;掌握定时/计数器工作在定时和计数方式下的电路连接和编程方法等。实践训练可按照下面给出的步骤进行。

1. 根据例 6-1 电路,编写程序以 50 Hz 的频率循环点亮 LED 发光管,并能够通过开关 S1, S2 调整 LED 的发光时间,按动开关 S1,频率以 1 Hz 为基础增大;按动开关 S2,频率以 1 Hz 为基础减少,观察发光管发光频率。循环点亮时间用定时器实现,程序编写用两种方法:查询和中断方式,比较两种方法的优缺点。
2. 完成例 8-4 例题。
3. 仿照例 8-5:  $\overline{\text{INT1}}$  引脚接入方波信号,利用定时/计数器 T1 测量  $\overline{\text{INT1}}$  引脚上出现的正脉冲宽度,并把机器周期数表示的结果利用发光管显示出来。发光管的连接电路按照图 6.4.1 连接。

## 第9章 AT89S52 单片机串行通信

AT89S52 具有一个全双工的串行口,可以通过编程设定为 4 种工作方式。和并行接口一样,串行接口也是单片机和外界通信的纽带,在数据传输、人机接口设计等方面起着重要作用。

本章的主要内容有:串行通信中的基本概念;RS232C 接口标准;波特率计算;串行口应用。

### 9.1 串行通信概述

计算机之间以及计算机与其他外部设备之间的信息交换称为数据通信。数据通信方式有两种,即并行通信和串行通信。

① 并行通信。并行通信时数据的各位同时传送。其优点是传送速度快;缺点是数据有多少位,就需要多少根数据线,在长距离传输的过程中,传输线过多是不经济的,并使系统的抗干扰能力下降。

② 串行通信。串行通信时数据的各位按一定的顺序逐位分时传送。它的突出优点是只需要一对数据线,大大降低了网络成本,特别适用于远距离通信;其缺点是传送速度较低。

#### 9.1.1 串行通信的实现

两个通信设备在串行线路上成功地实现通信必须解决以下几个问题:第一,串并转换和并串转换,即如何把要发送的并行数据串行化和把接收的串行数据并行化;第二,设备同步,即同步发送设备和接收设备的工作节拍,以确保发送数据在接收端被正确接收;第三,通信协议,即通信双方在通信前的约定,包括以何种方式通过什么样的速率发送,数据的帧格式如何等。

##### 1. 串并转换和并串转换

串行通信是将单片机内部的并行数据转换成串行数据,然后将其通过一条数据线传送出去;并将接收的串行数据再转换成并行数据送到计算机中(如单片机与打印机之间的数据传输)。这个过程一般通过移位寄存器来完成,74LS164 和 CD4094 是典型的串行输入并行输出移位寄存器;而 74LS165 和 CD4014 是典型的并行输入串行输出移位寄存器。

##### 2. 设备同步

进行串行通信的两台设备必须同步工作才能有效地检测通信线路上的信号变化,从而采样传送数据脉冲。设备同步对通信双方有两个共同的要求:一是通信双方必须采用统一的编码方式;二是通信双方必须能产生相同的传送速率。

采用统一的编码方法确定了一个字符的表达形式以及位发送顺序和位串长度等,当然还包括统一的逻辑电平规定,即电平信号高低与逻辑 1 和逻辑 0 的固定对应关系。

通信双方只有产生相同的传送速率才能确保设备同步,这就要求发送设备和接收设备采用相同频率的时钟。发送设备在统一的时钟脉冲下发送数据,接收设备才能正确检测出与时钟脉冲同步的数据信息。

### 3. 通信协议

通信协议是对数据传送方式的规定,包括数据格式、数据位定义、发送数据速率等。通信双方只有在遵从统一的通信协议的前提下,通信过程才能够正确地进行。异步通信协议一般包含以下几方面的内容。

#### (1) 起始位

通信线路上没有数据传送时一般处于逻辑 1 的状态。当发送设备有数据发送时,它首先要发出一个逻辑 0 信号,这个逻辑低电平就是起始位。当接收设备检测到这个低电平后,就开始准备接收数据位信号。起始位所起的作用就是设备同步,通信双方必须在传送数据位前协调同步。

#### (2) 数据位

数据位的个数依据通信双方的约定可以是 5、6、7 或 8 位。这些数据位被接收到移位寄存器中,构成传送数据字符。在字符数据传送过程中,数据位一般是从最低有效位开始发送,直至最高位发送结束。

#### (3) 奇偶校验位(或可编程第 9 位)

奇偶校验位的主要作用是用于差错控制,校验所接收数据的正确性。通信双方必须选择相同的校验方式。

#### (4) 停止位

在奇偶位或数据位(当无奇偶校验时)之后发送的是停止位。它是一个字符数据的结束标志,可以是 1 位、1.5 位或 2 位高电平。接收设备收到停止位之后,通信线路上便又恢复逻辑 1 状态,等待下一个起始位的到来。

#### (5) 波特率

波特率的设置主要是为了设定数据的发送速度。通信线上传送的所有位信号都保持一定的持续时间,以便接收端能够正确接收,而每一位信号持续时间是由数据传送速度决定的。每秒钟传送二进制位的个数,称为波特率。如果数据以每秒 9 600 个二进制位在通信线上传送,那么传送速度就是 9 600 Baud,记为 9 600 bit/s。

## 9.1.2 串行通信的通信方式

在串行通信中,发送方和接收方通过一根传输线连接起来,发送方把要发送的 1、0 数据转换成高低电平放到传输线上,经过一定的延迟之后,接收方采样这条传输线,并把采样到的高低电平转换成 1、0 数据。由于信息在一个方向上传输只占用一根传输线,而这根线上既传送数据(即通信双方真正要传送的数据信息),又传送联络信号,例如起始位、停止位等,为区分传送的信息中哪一部分是联络信号,哪一部分是数据,就必须对串行通信的联络信号与数据给予明确的规定。串行通信有同步通信和异步通信两种基本通信方



式,不同的通信方式对联络信号和数据有着不同的规定。

### 1. 同步通信

同步通信的基本特征是发送和接收时钟保持严格同步。采用同步通信时,将许多字符组成一个信息组,这样字符可以一个接一个地传输,但是在每组信息(通常称为帧)的开始要加上同步字符,在没有信息要传输时,要填上空字符,因为同步传输不允许有间隙,也不存在起始位和停止位,仅在数据块传输开始时用同步字符来指示。因此同步通信技术难度较大,但其通信速度较高,而且误码率较低。

同步通信中使用的数据格式根据所采用的控制规程(通信双方就如何交换信息所建立的一些规定和过程称为通信控制规程)又可分为面向字符型和面向位(比特)型两种。

面向字符型的数据格式又有单同步、双同步、外同步之分。

① 单同步:发送方先传送一个同步字符,再传送数据块,接收方检测到同步字符后接收数据。

② 双同步:发送方先传送两个同步字符,再传送数据块,接收方检测到同步字符后接收数据,如图 9.1.1 所示。

③ 外同步:用一条专用线来传送同步字符,以实现收发双方同步操作。

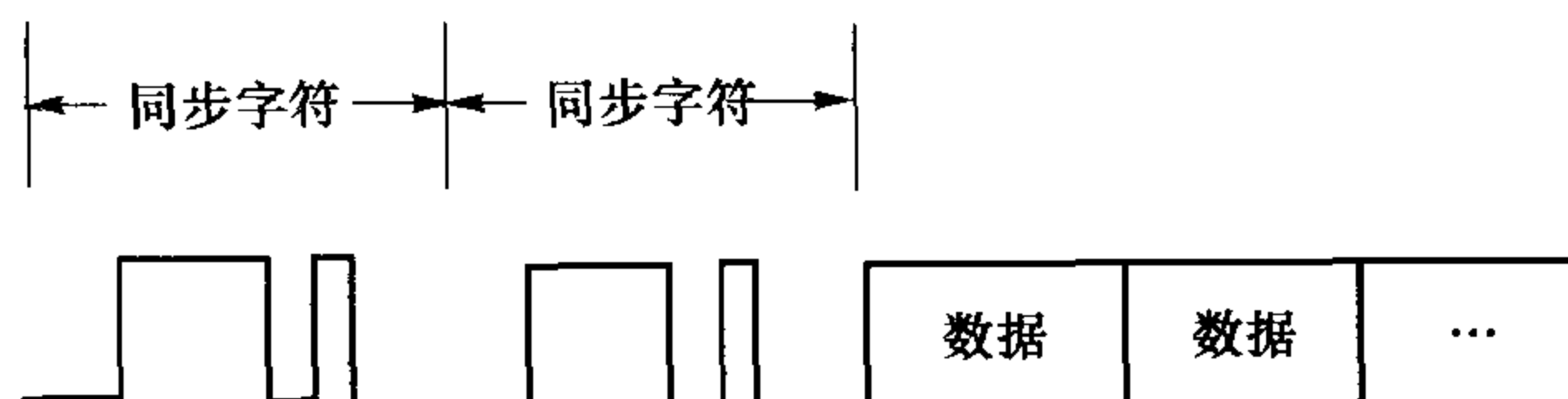


图 9.1.1 面向字符双同步数据帧格式

3 种同步方式,均以两个字节的冗余检验码 CRC 作为一帧信息的结束。

面向位型的数据格式如下。

根据 IBM 的同步数据链路控制规程 SDLC,SDLC 数据帧格式如图 9.1.2 所示。

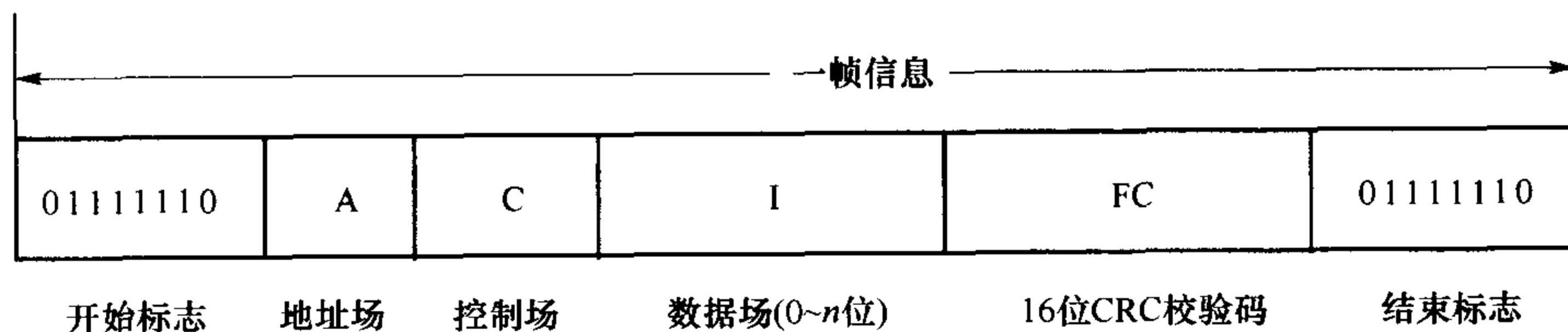


图 9.1.2 SDLC 数据帧格式

在同步通信中,要求用时钟来实现发送端与接收端之间的同步,接收和发送时钟对于收/发双方之间的数据传送达到同步是至关重要的。在发送方,一般都是在发送时钟的下降沿将数据串行移位输出;在接收方,一般都是在接收时钟的上升沿将数据串行移位输入。

### 2. 异步通信

异步通信不需要同步字符,也不需要发送设备保持数据块的连续性。可以准备好一个发送一个,但要发送的每一字符都必须先按照通信双方约定好的格式进行格式化,在其

前后分别加上起始位和停止位,用以指示每一字符的开始和结束。正是由于每一字符都包含有起始位和停止位,因此,异步通信的传输效率不如同步通信的效率高。但对接收与发送时钟的要求可以低一些。当第 8 位到来时,接收时钟稍微偏离发送时钟,只要不偏离太大,就不会影响字符的正确接收。异步通信的帧格式如图 9.1.3 所示。

在帧格式中,一个字符由 4 部分组成:起始位、数据位、奇偶校验位和停止位。传送一个字符总是从传送 1 位起始位(0)开始,接着传送字符本身(5~8 位),传送字符从最低位开始,逐位传送,直至传送最高位,接着传送奇/偶校验位,最后传送 1 位或 1.5 位或 2 位停止位(1)。从起始位开始到停止位结束,构成一帧信息。一帧信息传送完毕后,可传送不定长度的空闲位(1),作为帧与相邻帧之间的间隔,也可以没有空闲位间隔。

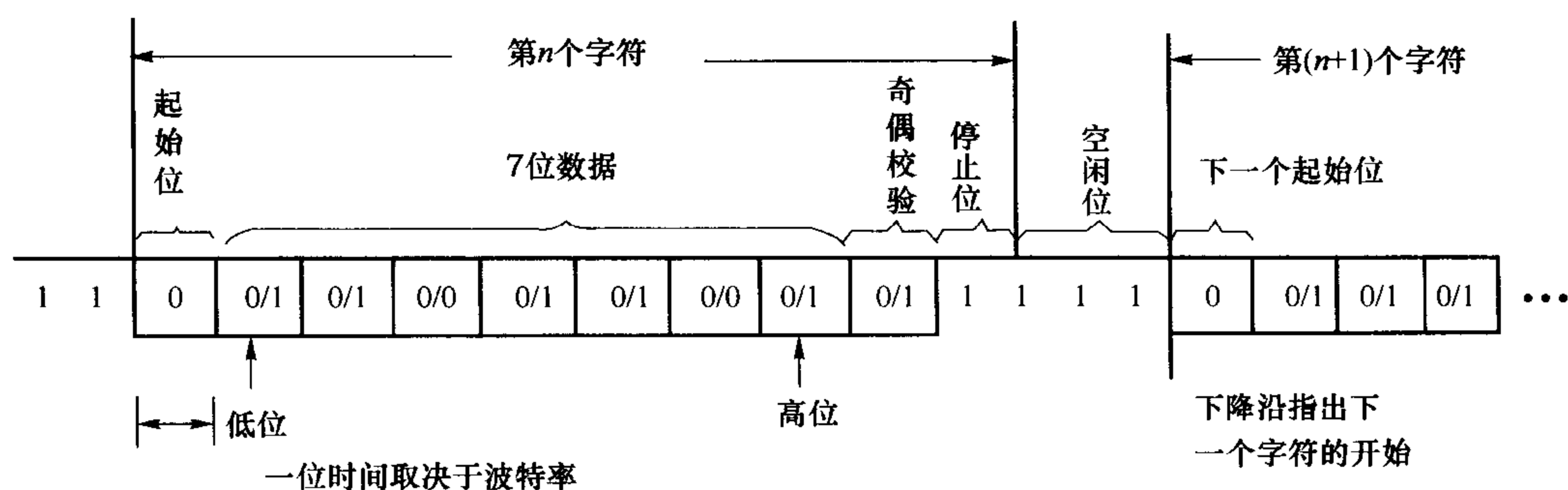


图 9.1.3 异步通信数据的帧格式

起始位用来通知接收设备开始接收数据。在线路空闲的时候应保持为 1,接收设备不断检测线路的状态,若连续为 1 后又检测到一个 0,就认为是新一帧数据的开始。起始位还被作为同步接收端的时钟,以保证以后的接收能正确进行。

数据位是数据通信中需要传送的信息,它可以是 5~8 位。

奇偶校验位占用 1 位,根据通信双方的通信协议进行设定。

停止位用来表示通信过程中一帧数据的结束,用 1 来表示。停止位可以是 1 位、1.5 位或者 2 位。计算机接收到停止位后,表示上一帧数据已经传输完成,同时为下一帧数据的接收作准备。

### 9.1.3 串行通信的传输方式

串行通信的传输方式可以分为单工、半双工及全双工 3 种。

#### 1. 单工通信

单工通信是指数据只能单方向传输的工作方式,因此只占用一个信道(信道是指以传输媒质为基础的信号通道)。广播、遥控、遥测、无线寻呼等属于单工通信。

#### 2. 半双工通信

半双工通信是指通信双方都能交替地进行双向数据传输,但两个方向的数据传输不能同时进行。例如,使用同一载波频率的对讲机、收发报机等都是半双工的通信方式。

### 3. 全双工通信

全双工通信是指通信双方可同时进行数据收发的工作方式。一般情况下全双工通信的信道必须是双向信道。普通电话、手机都是最常用的全双工通信方式,计算机之间的高速数据通信也是这种方式。

## 9.2 RS232C 标准总线及通信设计

RS232C 是由美国电子工业协会(EIA, Electronic Industry Association)制定的用于串行通信的标准通信接口,包括按位串行传输的电气和机械方面的规定,适用于短距离或带调制解调器的通信场合,利用它可以很方便地把各种计算机、外围设备、测量仪器等有机地连接起来,进行串行通信。为了提高数据传输速率和通信距离,EIA 还公布了 RS422,RS423 和 RS485 等串行总线接口标准。

### 9.2.1 RS232C 接口的引脚描述

RS232C 标准规定接口有 25 根连线,D 型插头和插座,采用 25 芯引脚或 9 芯引脚的连接器,RS232C 标准接口如图 9.2.1 所示。

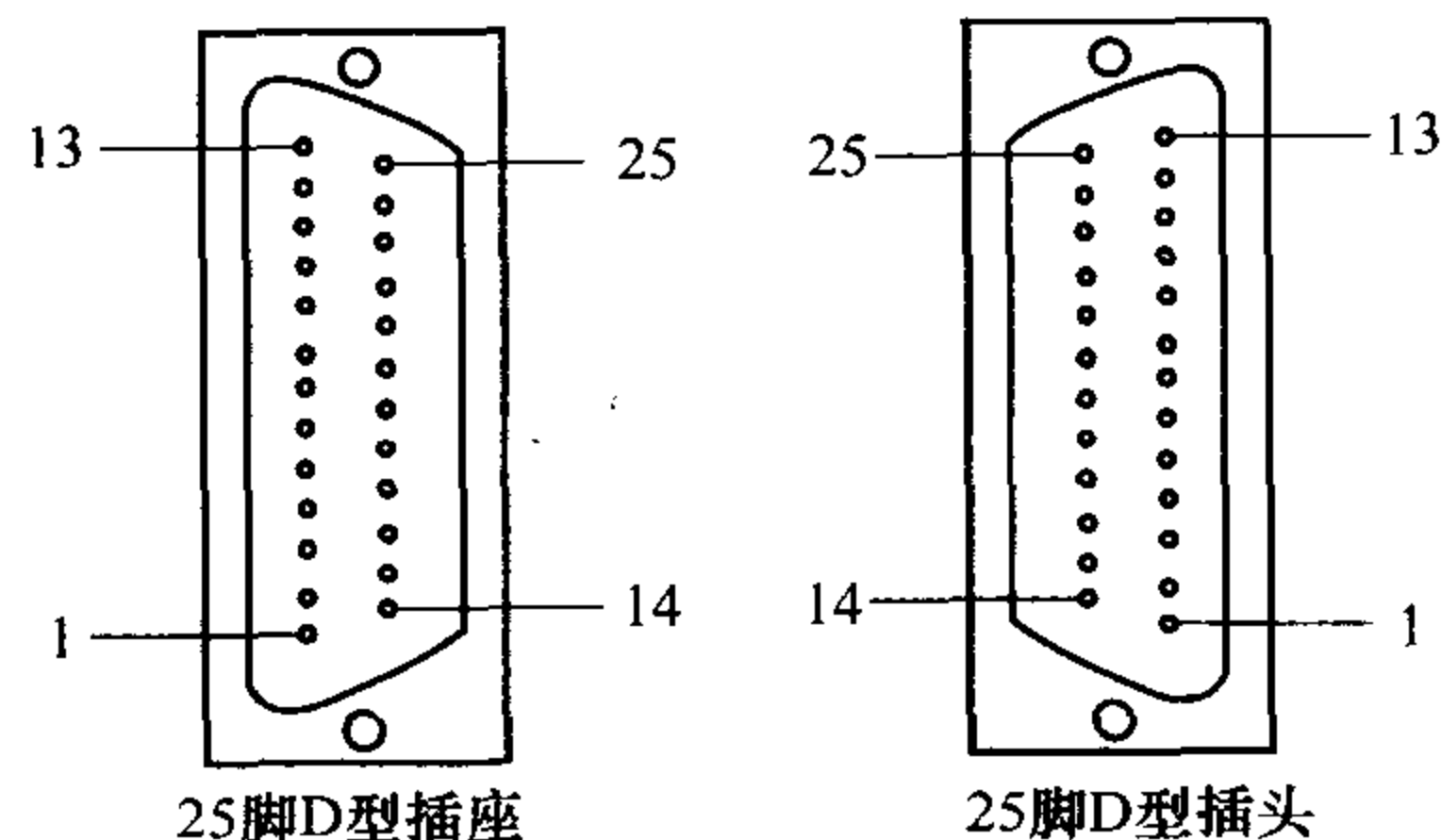


图 9.2.1 RS232C 标准接口图

虽然 RS232C 标准规定接口定义了 25 条连线,但通常只有以下 9 个信号经常使用,其对应关系如表 9.2.1 所示。

TXD:发送数据,输出。

RXD:接收数据,输入。

RTS:请求发送,输出。这是数据终端设备(以下简称 DTE)向数据通信设备(以下简称 DCE)提出发送要求的请求线。

CTS:允许发送,输入。这是 DCE 对 DTE 提出的发送请求做出的响应信号。当 CTS 在接通状态时,就是通知 DTE 可以发送数据了。当 RTS 在断开状态时,CTS 也随之断开,以备下一次应答过程的正常进行;当 RTS 在接通状态时,只有当 DCE 进入发送状态时,即 DCE 已准备接收 DTE 送来的数据进行调制,并且 DCE 与外部线路接通时,CTS 才处于接通状态。

表 9.2.1 9 针连接器和 25 针连接器间的对应关系

引脚描述	9 针连接器	25 针连接器
DCD	1	8
RXD	2	3
TXD	3	2
DTR	4	20
GND	5	7
DSR	6	6
RTS	7	4
CTS	8	5
RI	9	22

DSR:数据通信设备准备就绪,输入。它反映了本端数据通信设备当前的状态。当此线在接通状态时,表明本端 DCE 已经与信道连接上了且并没有处在通话状态或测试状态,通过此线,DCE 通知 DTE,DCE 准备就绪。DSR 也可以作为对 RTS 信号的响应,但 DSR 线优先于 CTS 线成为接通态。

GND:地。

DCD:接收线路信号检测,输入。这是 DCE 送给 DTE 的线路载波检测线。Modem 在连续载波方式工作时,只要一进入工作状态,将连续不断地向对方发送一个载波信号。每一方的 Modem 都可以通过对这一信号的检测,判断线路是否接通、对方是否正在工作。

DTR:数据终端准备就绪,输出。如果该线处于接通状态,DTE 通知 DCE,DTE 已经做好了发送或接收数据的准备,DTE 准备发送时,本设备是主动的,可以在准备好时,将 DTR 线置为接通状态。如果 DTE 具有自动转入接收的功能,当 DTE 接到振铃指示信号 RI 后,就自动进入接收状态,同时将 DTR 线置为接通状态。

RI:振铃检测,输入。当 DCE 检测到线路上有振铃信号时,将 RI 线接通,传送给 DTE,在 DTE 中常常把这个信号作为处理机的中断请求信号,使 DTE 进入接收状态,当振铃停止时,RI 也变成断开状态。

## 9.2.2 RS232C 接口的具体规定

### 1. 电气性能规定

(1) 在 TXD 和 RXD 线上,RS232C 采用负逻辑

逻辑正(即数字 1) =  $-15 \sim -3$  V;

逻辑负(即数字 0) =  $+3 \sim +15$  V。

(2) 在联络控制信号线上(如 RTS、CTS、DSR、DTR、RI、DCD 等)

ON(接通状态) =  $+3 \sim +15$  V;

OFF(断开状态) =  $-15 \sim -3$  V。



## 2. 传输距离

RS232C 标准适用于 DCE 和 DTE 之间的串行二进制通信,最高的数据速率为 19.2 kbit/s,在使用此波特率进行通信时,最大传送距离在 20 m 之内。降低波特率可以增加传输距离。

### 9.2.3 RS232C 接口的典型应用

RS232C 总共定义了 25 根信号线,但在实际应用中,使用其中多少根信号线并无约束,也就是说,对于 RS232C 标准接口的使用是非常灵活的,实际通信中经常采用 9 针接口进行数据通信,9 针串口插头和插座实物如图 9.2.2 所示。下面给出 3 种典型的使用 RS232C 的连接方式,如图 9.2.3~图 9.2.5 所示。

图 9.2.3 是两个 DTE 之间进行 RS232C 串行通信的典型连接,但这种信号线的连接方式不是唯一的。在图中连接方式下,信号传送的过程是:首先发送方将 RTS 置为接通,向对方请求发送,由于接收方的 DSR 和 DCD 均和发送方的 RTS 相连,故接收方的 DSR 和 DCD 也处于接通状态,分别表示发送方准备就绪和告知接收方对方请求发送数据。当接收方准备就绪,准备接收数据时,就将 DTR 置为接通状态,通知发送方接收方准备就绪,由于发送方的 CTS 接到接收方的 DTR,故发送数据,接收方从 RXD(接到发送方 TXD)接收数据。如果接收方来不及处理数据,接收方可暂时断开 DTR 信号,迫使对方暂停发送。当发送方数据发送完毕,便可断开 RTS 信号,接收方的 DSR 和 DCD 信号状态也就处于断开状态,通知接收方,一次数据传送结束。如果双方都是始终在就绪状态下准备接收数据,连线可减至 3 根,即 TXD、RXD 和 SGND。

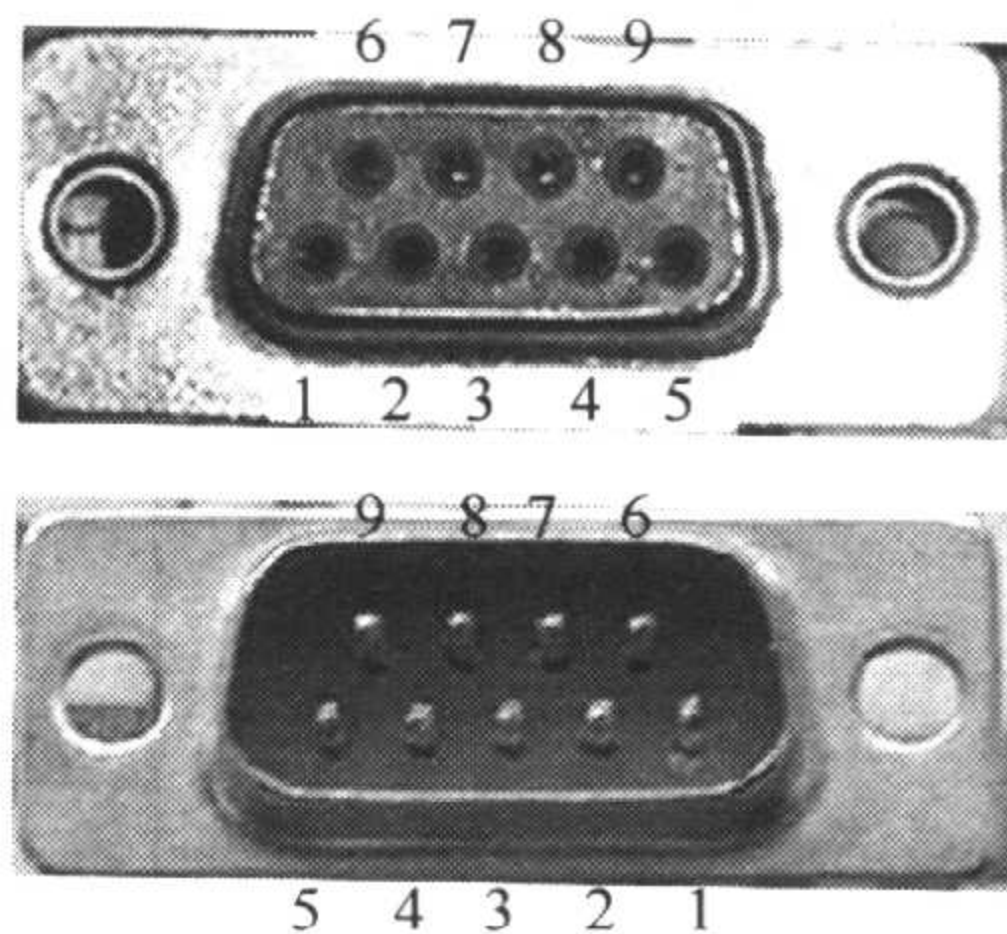


图 9.2.2 9 针串口插头和插座实物图

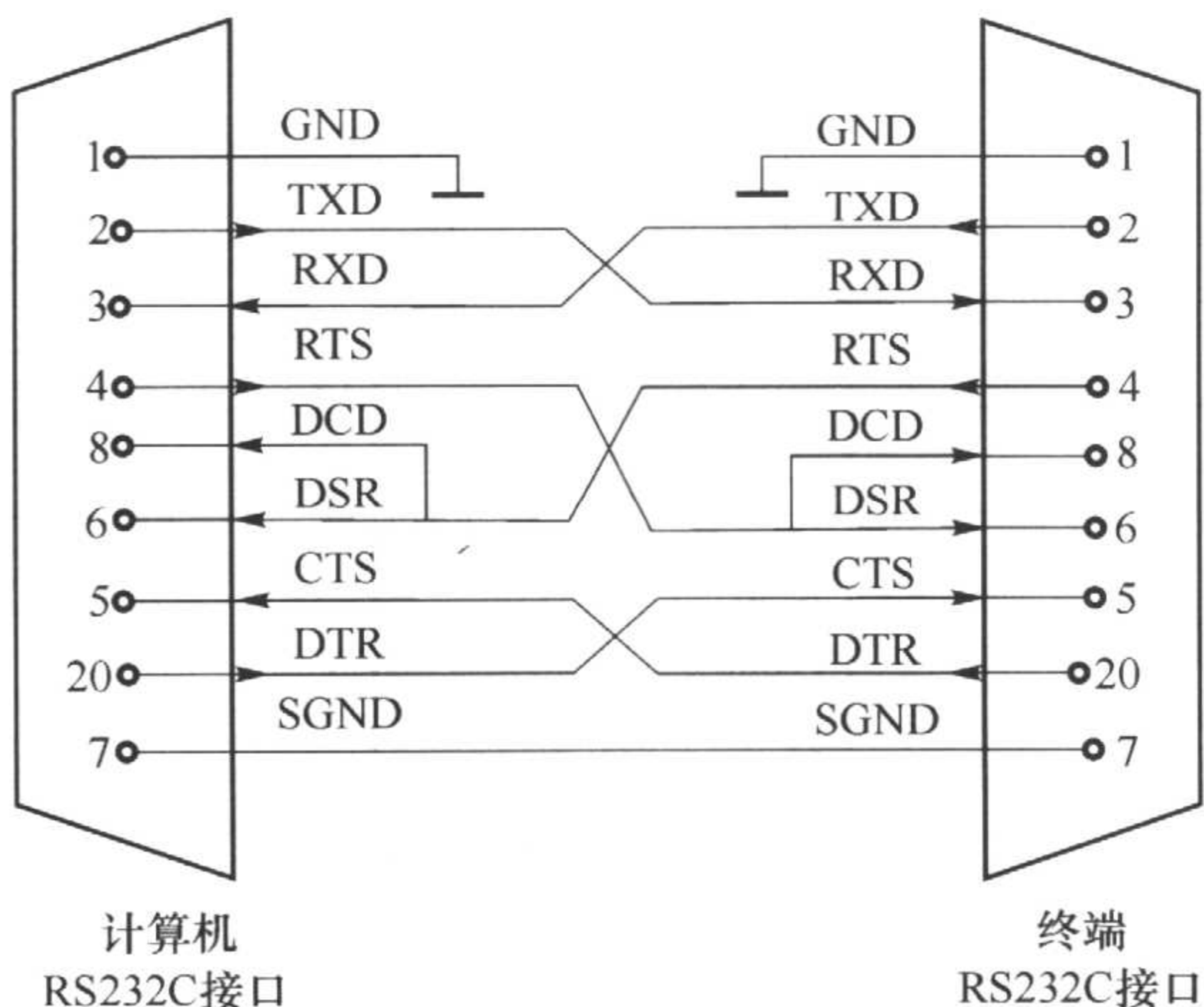


图 9.2.3 两个 RS232C 设备通信连线图

图 9.2.4 是单片机与计算机之间通信的连接示意图。由于单片机输入、输出为 TTL 电平,而 PC 配置的是 RS232C 标准串行接口,两者的电气特性不一致,因此,要完成 PC 与单片机之间的数据通信,必须进行电平转换。MC1488 负责将 TTL 电平转换为 RS232C 电平;而 MC1489 则是把 RS232C 电平转换为 TTL 电平。PC 输出的电平信号经过 MC1489 转换成 TTL 电平信号,送到单片机的 RXD 端;单片机的串行发送引脚 TXD 输出的 TTL 电平信号经过 MC1488 电平转换器转换成 PC 可接收的电平信号,接到 PC 的 RXD 端。

当电路的传输距离较远时,即使使用双绞线也容易引起干扰,所以在 MC1488 的输出端最好外加电容滤波,电容的值通常为  $0.01\ \mu\text{F}$ 。此电路结构简单,可靠性好。它的缺点是需提供  $\pm 12\ \text{V}$  和  $+5\ \text{V}$  双电源,因而存在着一定的局限性。

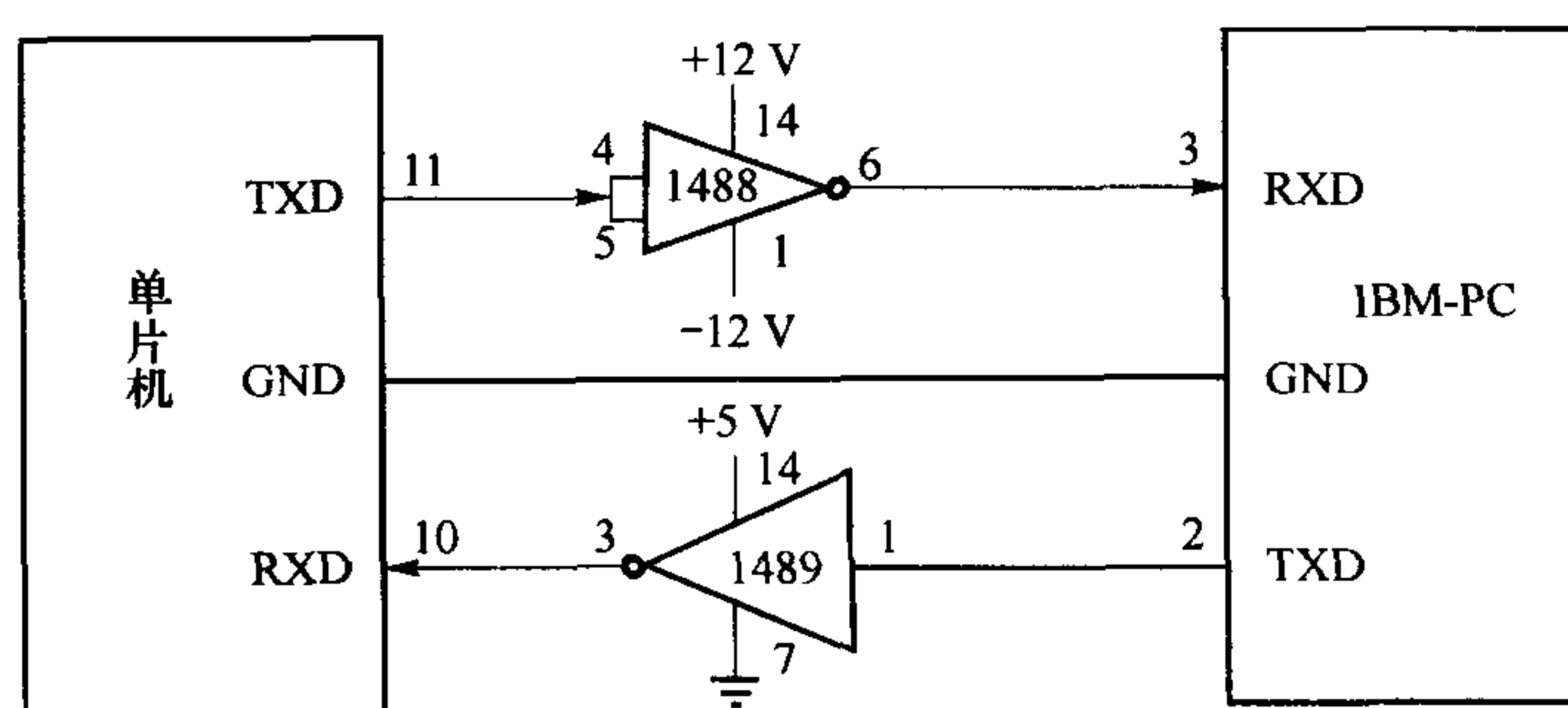


图 9.2.4 单片机和 PC 采用 MC1488 和 MC1489 通信连线图

图 9.2.5 是单片机与计算机之间采用 MAX232 芯片通信的连接示意图。MAX232 是 MAXIM 公司生产的包含两路接收器和驱动器的 IC 芯片,其芯片内部具有电源电压变换器,可以把输入的  $+5\ \text{V}$  电压变换成为 RS232C 输出电平所需要的  $\pm 10\ \text{V}$ 。此芯片只需  $+5\ \text{V}$  供电,因此它的适应性更强。

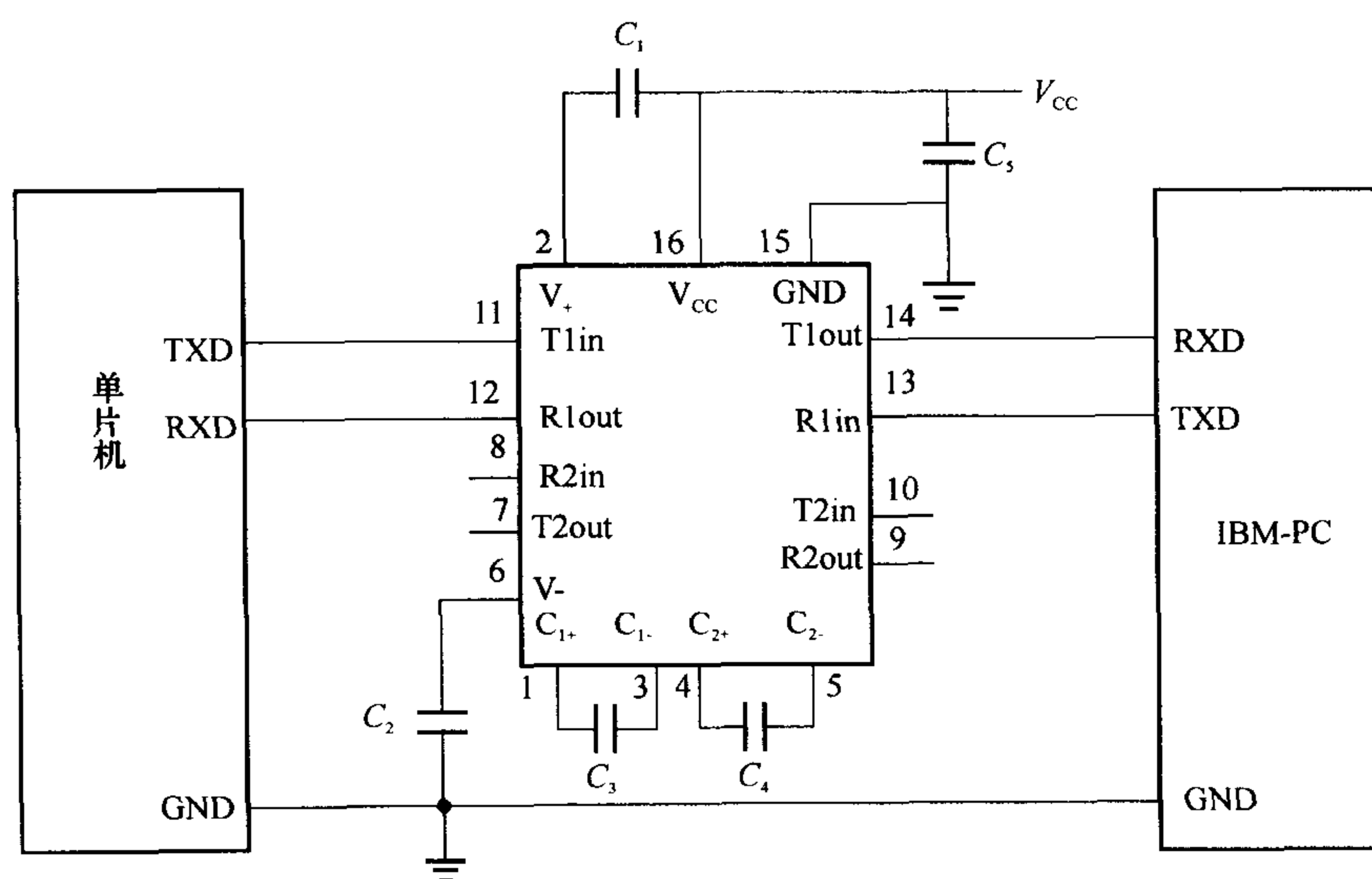


图 9.2.5 单片机和 PC 采用 MAX232 通信连线图

实际应用中可采用 MAX232 芯片中两路发送接收中的任意一路作为接口。要注意的是其发送和接收的引脚要互相对应。图 9.2.5 中采用 T1in 引脚接单片机的 TXD, 则 PC 的 RS232 接收端一定要对应接到 T1out 引脚。同时, R1out 接单片机的 RXD 引脚, 对应 PC 的 TXD 应接到 MAX232 中的 R1in 引脚。C<sub>1</sub>、C<sub>2</sub>、C<sub>3</sub>、C<sub>4</sub> 和 V<sub>+</sub>、V<sub>-</sub> 主要用于电源变换, 其中的电容选用钽电容效果比较好, 4 个电容的典型值一般为 0.1  $\mu$ F。

### 9.3 AT89S52 串行通信接口

AT89S52 单片机具有一个全双工串行口, 既可以工作在同步模式, 又可以工作在异步的 UART(通用异步收发器)模式, 能方便地构成双机、多机串行通信接口。

#### 9.3.1 串行口的控制

串行口的工作模式设定和控制由特殊功能寄存器 SCON 和 PCON 中的 SMOD 来完成, 发送或接收的数据存储在 SBUF 中, 系统复位后, SCON 和 PCON 的值被清 0。而 SBUF 中的值不确定。

##### 1. 串行数据缓冲器 SBUF

串行数据缓冲器 SBUF 的地址是 99H。实际上它对应着两个缓冲器, 一个是串行发送缓冲器, 另一个是串行接收缓冲器, 两个缓冲器共用同一个物理地址。CPU 执行写 SBUF 命令(MOV SBUF, A), 实质上把数据写入串行发送缓冲器, 而 CPU 执行读 SBUF 命令(MOV A, SBUF), 实质上就是从串行接收缓冲器中读取数据。另外, 接收缓冲器是双缓冲的, 它能够在接收到的第一帧数据从接收寄存器被读出之前就开始接收第二帧数据, 以避免 CPU 未能及时将上一帧数据取走而产生两帧数据重叠的问题。发送缓冲器是单缓冲的, 主要是为了实现最大的传输速率, 因为发送时 CPU 是主动的, 所以不会产生重叠的问题。

##### 2. 串行口控制寄存器 SCON

串行口的工作模式设定和基本控制操作通过设定串行口控制寄存器 SCON 完成, 串行口控制寄存器 SCON 除可进行字节寻址外, 还可以进行位寻址, 其各位定义及格式如图 9.3.1 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
SCON (98H)	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H

图 9.3.1 控制寄存器 SCON 的位定义

SM0, SM1: 串行口工作模式选择位。对应着串行口的 4 种工作模式, 如表 9.3.1 所示。其中  $f_{osc}$  是振荡器频率。



表 9.3.1 串行口工作模式

SM0	SM1	工作模式	说明	波特率
0	0	模式 0	同步移位寄存器	$f_{osc}/12$
0	1	模式 1	8 位 UART	由定时器控制
1	0	模式 2	9 位 UART	$f_{osc}/32$ 或 $f_{osc}/64$
1	1	模式 3	9 位 UART	由定时器控制

SM2:多机通信控制位。

- 在模式 0 下, SM2 应为 0, 不用 TB8 和 RB8。
- 在模式 1 下, 当 SM2=0 时, RB8 是接收到的停止位, 由该位可判断一帧数据是否发送完成; 当 SM2=1 时, 则只有收到有效的停止位才会激活 RI。
- 在模式 2 和模式 3 的情况下, TB8 是发送的第 9 位数据, 可以用软件置位或清 0, RB8 是接收到的第 9 位数据。而且, 当 SM2=1 时, 同时接收到的第 9 位数据如果是 0, 则不激活 RI, 就是利用这一特点实现多机通信。

REN: 串行接收允许位。由软件置位或清 0, REN=1, 允许接收; REN=0, 不允许接收。

TB8: 发送数据的第 9 位数据。由软件置位或清 0, 此位既可以用来作为奇偶校验位, 又可在多机通信应用中, 用来作为地址帧和数据帧的标志位。

RB8: 接收数据的第 9 位数据。此位与 TB8 位相对应。

TI: 发送中断标志。在一帧数据发送完时被置位。在模式 0 串行发送第 8 位数据结束或在其他方式下串行发送到停止位的开始时由硬件置位, 此标志可由软件查询。它同时也向 CPU 申请中断, TI 置位意味着向 CPU 提供“发送缓冲器 SBUF 已空”的信息, CPU 可以准备发送下一帧数据。串行口发送中断被响应后, RI 不会自动清 0, 必须由软件清 0。所以一帧数据发送完成后, 必须清 0 TI 标志(CLR TI), 以便检查下一帧数据的发送完成情况。

RI: 接收中断标志。在接收完一帧有效数据后被置位。在模式 0 接收完第 8 位数据或在其他方式下接收到停止位的中间时由硬件置位, 此标志可由软件查询。它同时也向 CPU 申请中断, RI 置位意味着向 CPU 提供“接收缓冲器 SBUF 已满”的信息, 要求 CPU 把 SBUF 中的数据取走。串行口接收中断被响应后, RI 不会自动清 0, 必须由软件清 0。所以一帧数据接收完成后, 必须清 0 RI 标志(CLR RI), 以便正确接收下一帧数据。

串行发送中断标志 TI 和串行接收中断标志 RI 是同一个中断源, CPU 事先并不知道所产生的串行口中断到底是发送中断 TI 还是接收中断 RI 产生的中断请求, 所以在全双工通信时, 必须由软件来判断, 即根据 TI 或 RI 的状态, 判断是发送中断还是接收中断。

### 3. 电源控制寄存器 PCON

电源控制寄存器 PCON 中只有 SMOD(PCON. 7)位与串行口的工作有关, 如图 9.3.2 所示。需要注意的是, PCON 寄存器不能位寻址, 对于 SMOD 位的修改只能采用字节操作方式, 如 MOV PCON, #80H。

	D7	D6	D5	D4	D3	D2	D1	D0
PCON (87H)	SMOD	—	—	—	—	—	—	—

图 9.3.2 电源控制寄存器 PCON 中与串行口有关的位定义

SMOD:波特率加倍位。在串行口模式1、模式2和模式3时,波特率和 $2^{\text{SMOD}}$ 成正比,即当SMOD=1时,波特率提高一倍。系统复位后,SMOD=0。

### 9.3.2 串行口的工作模式

#### 1. 模式0

当SM0=0,SM1=0时,串行口工作在模式0,即同步移位寄存器模式,其结构如图9.3.3所示。串行口工作在模式0,其波特率是固定的,为 $f_{\text{osc}}/12$ ,数据由RXD(P3.0)端发送或接收,同步移位脉冲由TXD(P3.1)端输出,发送、接收的是8位数据,低位在先。

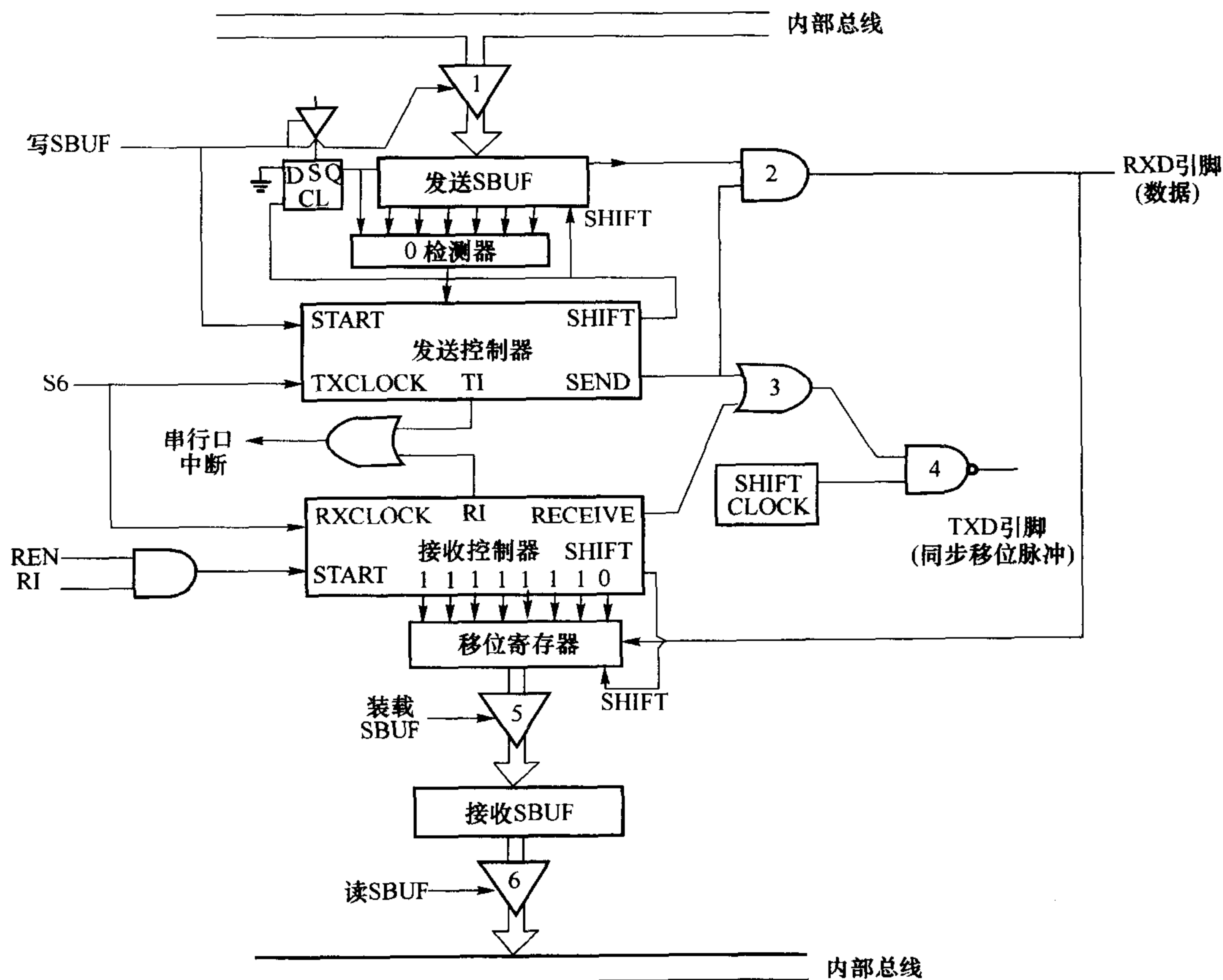


图 9.3.3 串行口模式 0 的结构原理图

#### (1) 发送

执行任何一条以SBUF为目的寄存器的指令(如MOV SBUF,A),则启动发送过程。发送的时序如图9.3.4所示的上半部分。

写SBUF命令执行过程中的写信号打开图9.3.3的三态门1,数据经由数据总线写入串行发送SBUF中,写信号同时启动发送控制器。经过一个机器周期,发送控制端

SEND 有效(高电平),打开与门 2、或门 3 和与非门 4,允许 RXD 引脚发送数据,TXD 引脚输出同步移位脉冲。每个机器周期的 S6P2 使输出移位寄存器的内容右移一位,左边补 0,发送数据缓冲器中的数据逐位串行输出。每一个机器周期从 RXD 引脚上发送一位数据,故波特率为  $f_{osc}/12$ 。S6 同时形成同步移位脉冲,一个机器周期从 TXD 引脚上输出一个同步移位脉冲。一帧数据发送完毕后,SEND 引脚恢复低电平状态,停止发送数据,同时发送控制器置位发送中断标志 TI,向 CPU 申请中断。

如要再次发送数据,必须用软件将 TI 清 0,并再次执行写 SBUF 命令。

## (2) 接收

在接收中断标志 RI 没有置位的情况下,将 REN(SCON. 4)置 1 则启动一次接收过程。此时 RXD 为串行数据接收端,TXD 依然输出同步移位脉冲。模式 0 的接收时序见图 9.3.4 的下半部分。

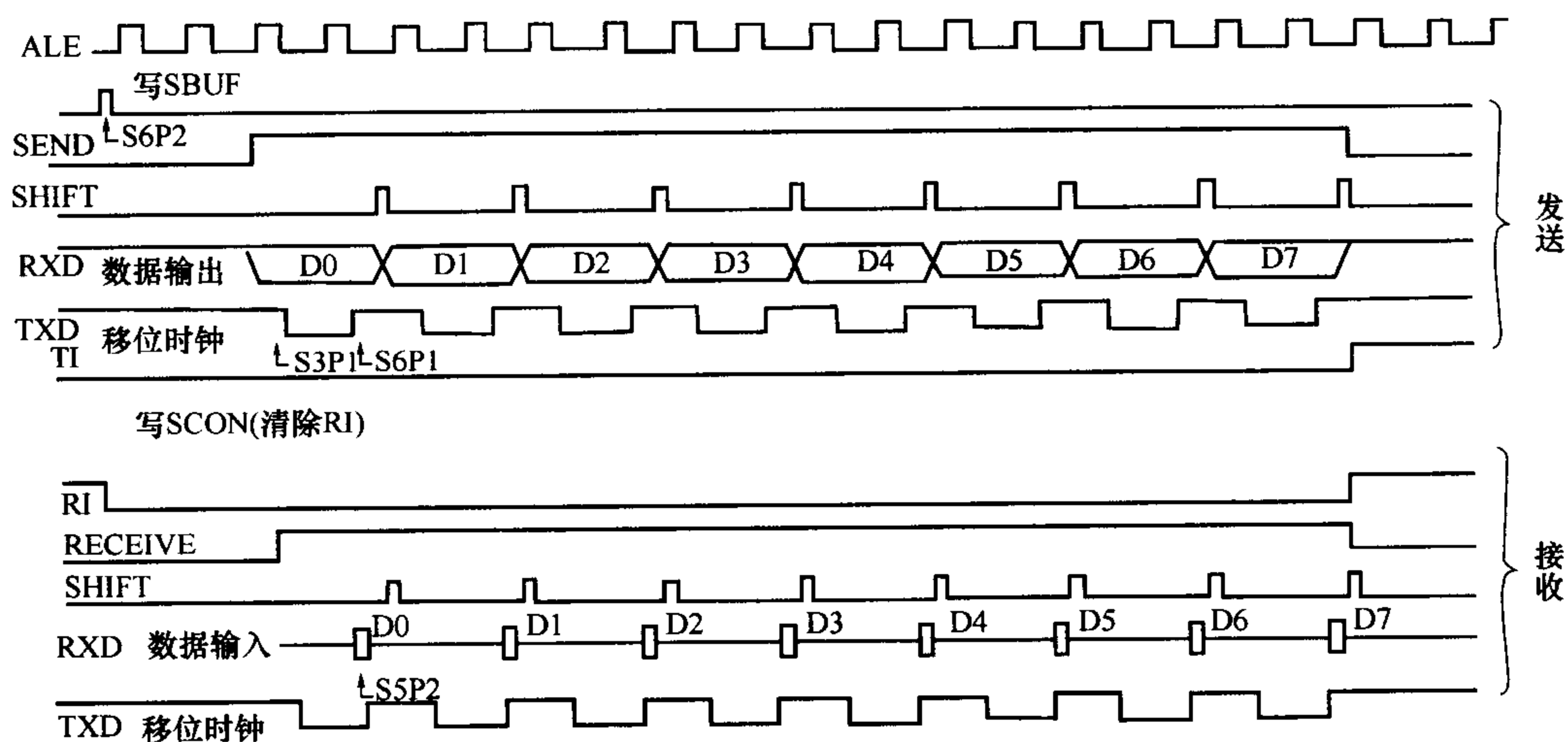


图 9.3.4 串行口模式 0 的工作时序图

REN 置位启动了接收控制器。经过一个机器周期,接收控制端 RECEIVE 有效,依次打开或门 3 和与非门 4,允许 TXD 输出同步移位脉冲。该脉冲控制外部设备逐位输入数据,波特率为  $f_{osc}/12$ 。在内部移位脉冲的作用下,RXD 上的串行输入数据逐位移入移位寄存器。当一帧数据全部移入移位寄存器后,接收控制器使 RECEIVE 失效,停止输出移位脉冲,还发出“装载 SBUF”信号,打开三态门 5,将 8 位数据并行送入接收缓冲器 SBUF 保存。与此同时,接收控制器硬件置位接收中断标志 RI,向 CPU 申请中断。CPU 响应中断后,用软件使 RI 清 0,使移位寄存器开始接收下一帧数据。CPU 执行读 SBUF 命令(如 MOV A, SBUF),读信号打开三态门 6,数据经由内部总线进入 CPU。

## 2. 模式 1

当  $SM0=0, SM1=1$  时,串行接口工作于模式 1,即 8 位异步收发器。此时通过 TXD 发送或 RXD 接收,其结构图如图 9.3.5 所示。发送或接收的位为:1 位起始位(0),8 位数

据位,1 位停止位(1),每帧 10 位。波特率可变,由定时器的溢出率和 SMOD(PCON. 7) 决定。

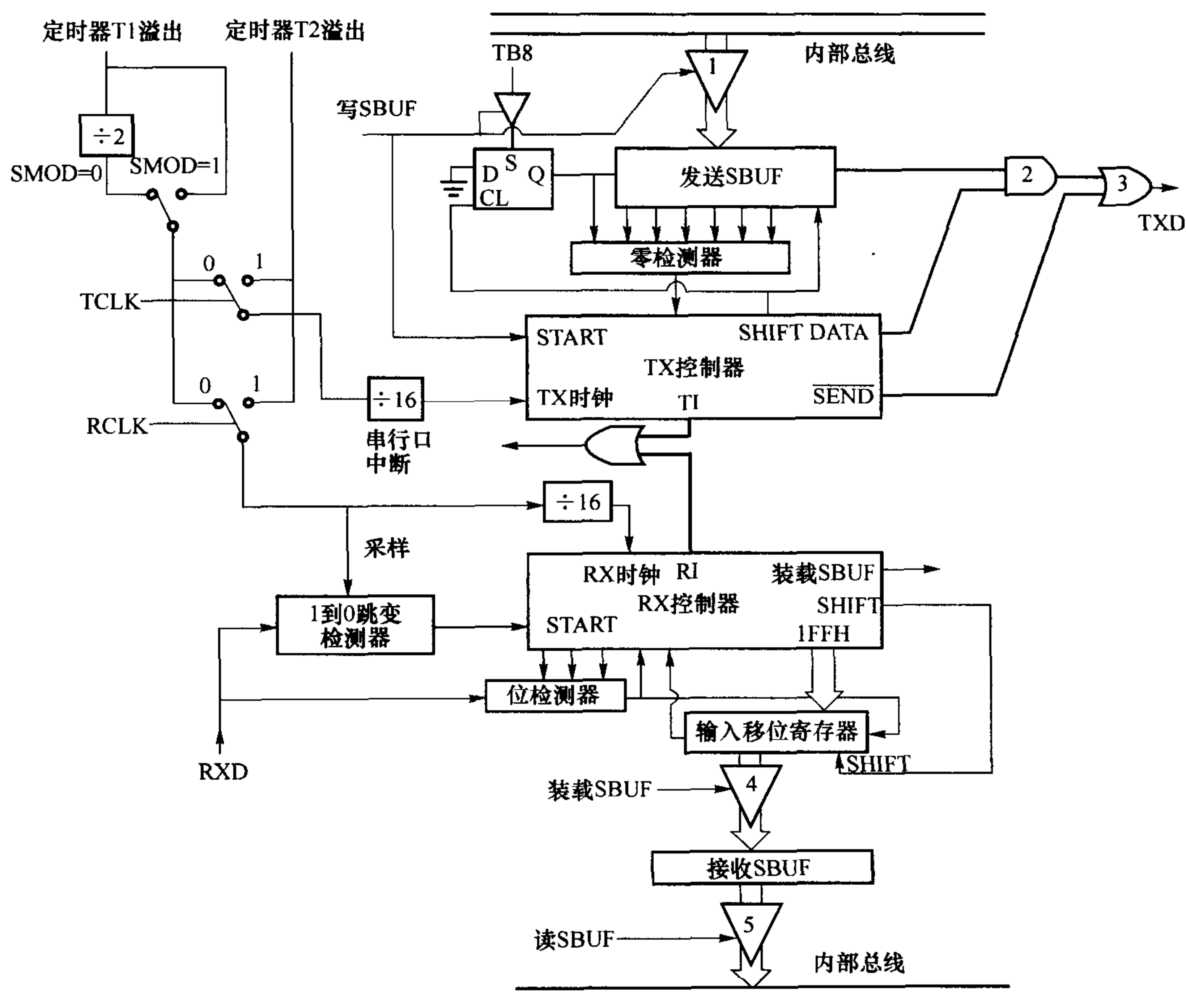


图 9.3.5 串行接口模式 1,3 的结构原理图

### (1) 发送

执行任何一条以 SBUF 为目的寄存器的指令时(如 MOV SBUF, A),则启动发送过程。发送的时序见图 9.3.6 的上半部分。

写 SBUF 命令打开三态门 1,并行数据传送至发送缓冲器中,同时启动 TX 控制器,经过一个机器周期, TX 控制器的  $\overline{\text{SEND}}$ 、DATA 相继有效,通过输出控制门 2 和 3 从 TXD 上逐位输出一帧信息。一帧信息发送完后,  $\overline{\text{SEND}}$ 、DATA 失效, TX 控制器硬件置位发送中断标志 TI,向 CPU 申请中断。

### (2) 接收

模式 1 的接收时序见图 9.3.6 的下半部分。

允许接收位 REN 被置位,接收器就开始工作。跳变检测器以波特率 16 倍的速率采样 RXD 引脚上的电平。当采样到从 1 到 0 的负跳变(起始位)时,启动接收寄存器接收数

据。由于发送、接收双方各自使用自己的时钟,两者的频率总有少许差异。为了避免这种影响,控制器将 1 位传送时间等分成 16 份,位检测器在 7,8,9 三个状态(也就是在位信号中央)采样 RXD 3 次。而且,3 次采样值中至少两次相同的值被确认为数据,这是为了减小干扰的影响。如果起始位接收到的值不是 0,则起始位无效,复位接收电路。如果起始位是 0,则开始接收本帧其他各位数据。RX 控制器发出的内部移位脉冲将 RXD 上的数据逐位移入输入移位寄存器,当 8 位数据及停止位全部移入后,对所接收的数据进行数据判别:

- ① 如果  $RI=0, SM2=0$ , 接收控制器发出“装载 SBUF”信号,将 8 位数据装入接收缓冲器,停止位装入 RB8,并置位 RI,向 CPU 申请中断;
- ② 如果  $RI=0, SM2=1$ , 那么只有停止位为 1 时才发生上述动作;
- ③ 如果  $RI=0, SM2=1$  且停止位为 0, 所接收的数据就会丢弃;
- ④ 如果  $RI=1$ , 则所接收的数据在任何情况下都会丢弃。

无论出现哪一种情况,跳变检测器将继续采样 RXD 引脚上的负跳变(起始位),以便接收下一帧数据。

接收器采用移位寄存器和 SBUF 双缓冲结构,以避免在接收后一帧数据之前,CPU 尚未及时响应中断将前一帧数据取走,造成两帧数据重叠的问题。采用双缓冲结构后,前后两帧数据进入 SBUF 的时间间隔至少有 10 个位传送周期。在后一帧数据进入 SBUF 之前,CPU 有足够的时间将前一帧数据取走。

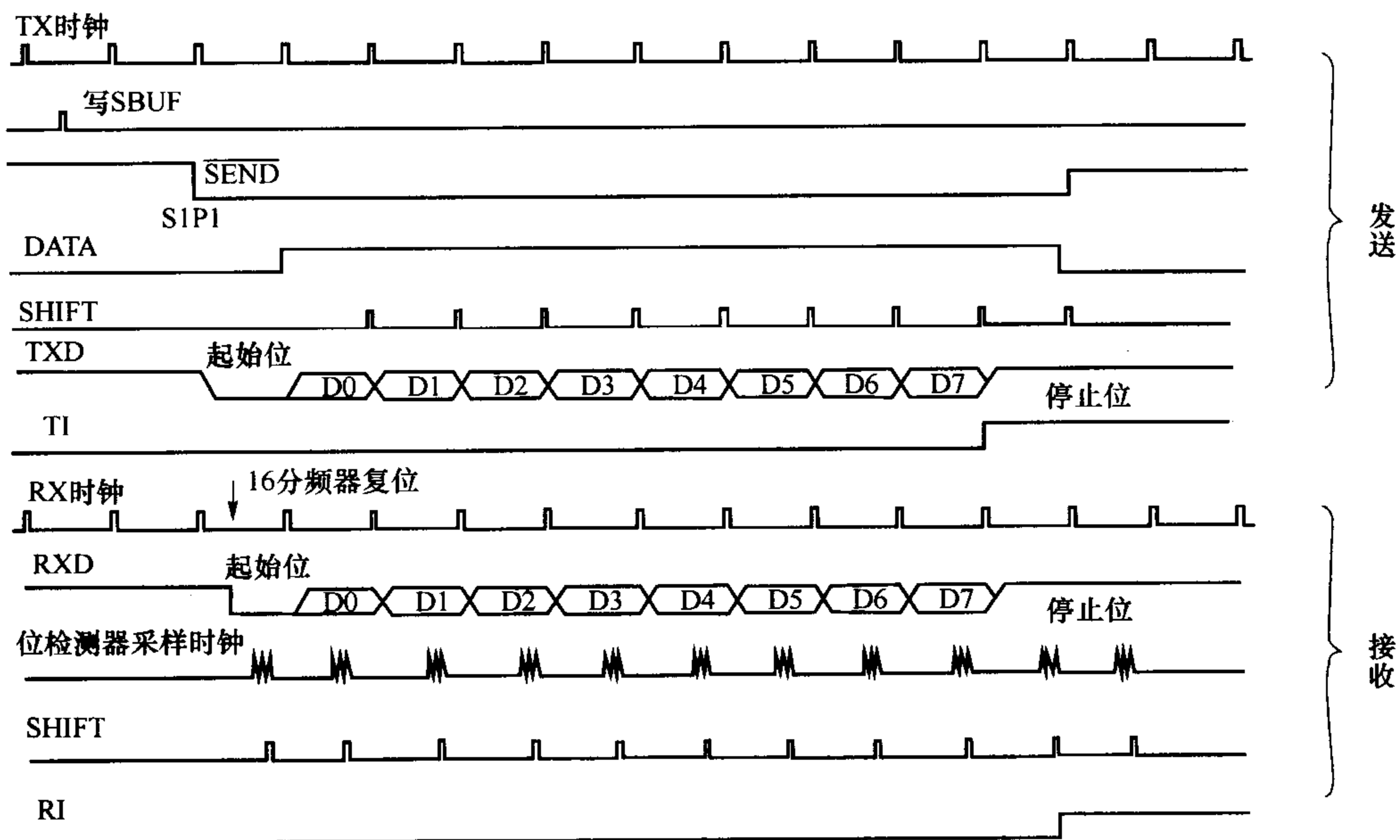


图 9.3.6 串行接口模式 1 工作时序图

### 3. 模式 2 和模式 3

当  $SM0=1, SM1=0$  时, 串行接口工作于模式 2, 即波特率固定的 9 位异步收发器, 其波特率为  $f_{osc}/32$  ( $SMOD=1$  时) 或  $f_{osc}/64$  ( $SMOD=0$  时); 而当  $SM0=1, SM1=1$  时, 串行接口工作于模式 3, 即波特率可变的 9 位异步收发器, 其波特率由定时器 1 或者定时

器 2 的溢出速率决定。这两种模式都是通过 TXD 发送和 RXD 接收。发送或接收的位为:1 位起始位(0),8 位数据位,可编程第 9 位和 1 位停止位,每帧 11 位。串行口工作在模式 2 的结构如图 9.3.7 所示。

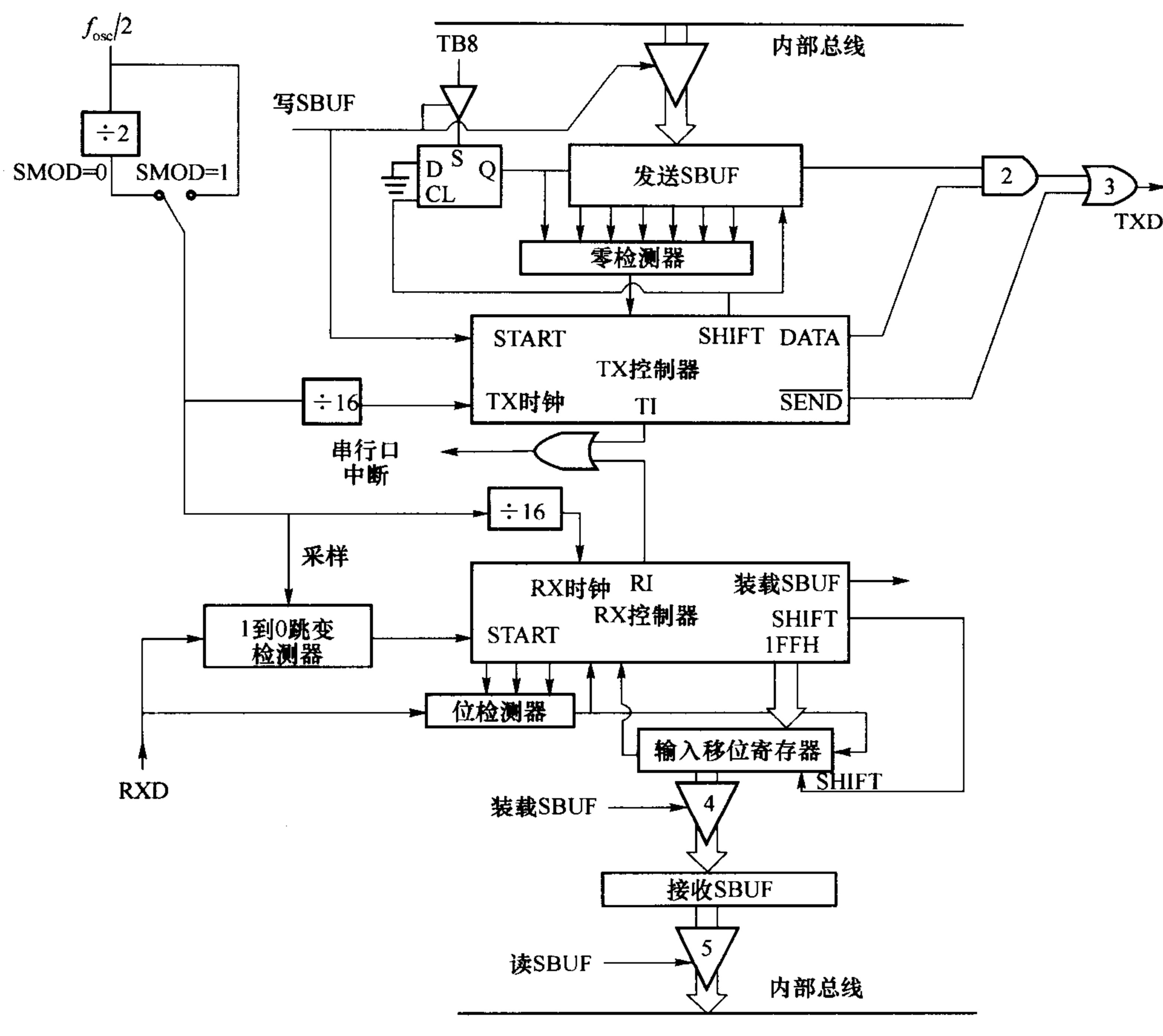


图 9.3.7 串行口工作在模式 2 的结构原理图

### (1) 发送

模式 2 和模式 3 发送时,数据从 TXD 引脚上输出,附加的第 9 位数据由 SCON 中的 TB8 位提供。CPU 执行一条写入 SBUF 的指令后立即启动发送器发送。发送完一帧数据后由硬件置位 TI,向 CPU 申请中断。

### (2) 接收

模式 2 和模式 3 在接收时和模式 1 类似,REN 置位后,跳变检测器不断对 RXD 引脚采样。当采样到负跳变后就启动接收控制器。位检测器对每位数据采集 3 个值,3 次采样值中至少两次相同的值被确认为数据。当第 9 位数据移入移位寄存器后,将 8 位数据装入 SBUF,第 9 位数据装入 RB8,并置位 RI,向 CPU 申请中断。

串行口工作在模式 2 和模式 3 的情况下其工作时序图与串行口工作在模式 1 的情况下的工作时序图类似,所不同的仅仅是模式 2 和模式 3 在模式 1 发送(接收)完 D7 位后还

要发送 TB8(接收 RB8),然后才是停止位。

与模式 1 相同,模式 2 和模式 3 也设置有数据辨别功能。当  $RI=0, SM2=0$  或  $RI=0, SM2=1$  时,接收到的第 9 位数据为 1,则接收到的数据送入 SBUF(接收缓冲器),第 9 位数据送入 RB8 并使  $RI=1$ ,向 CPU 请求中断。若两个条件都不满足,则接收的信息将被丢弃。

请注意在模式 1 中装入 RB8 的是停止位,而模式 2 和模式 3 中装入 RB8 的是可编程的第 9 位数据。

### 9.3.3 多机通信

#### 1. 多机通信原理

如前所述,串行口控制寄存器 SCON 中 SM2 位为模式 2 和模式 3 的多机通信使能位。当串行口以模式 2 和模式 3 接收时,若  $SM2=1$ ,此时仅当串行口接收到的第 9 位数据为 1 时,数据才装入 SBUF,并使中断标志  $RI=1$ ,向 CPU 发出中断请求;如果接收到的第 9 位数据为 0 时,则不产生中断标志,信息被丢弃。若  $SM2=0$ ,则接收到的第 9 位数据不论是 0 还是 1,都将接收到的数据装入 SBUF 中,并使中断标志  $RI=1$ 。应用串行口的这个特性,便可实现多机通信。一般利用第 9 位 TB8 作为地址/数据识别位,  $TB8=1$ ,表示发送的是地址帧;  $TB8=0$ ,则发送的是数据帧。

#### 2. 多机通信应用举例

设在一个多机系统中有 1 个主机和 3 个从机,其结构如图 9.3.8 所示。

设从机的地址分别定义为 04,05,06,从机系统由初始化程序将串行口编程为模式 2 接收,即 9 位异步通信方式,且置  $SM2=1, REN=1$ ,允许串行口中断。为了区分是数据信息还是地址信息,主机用第 9 位数据 TB8 作为地址/数据的识别位,地址帧的  $TB8=1$ ,数据帧的  $TB8=0$ 。

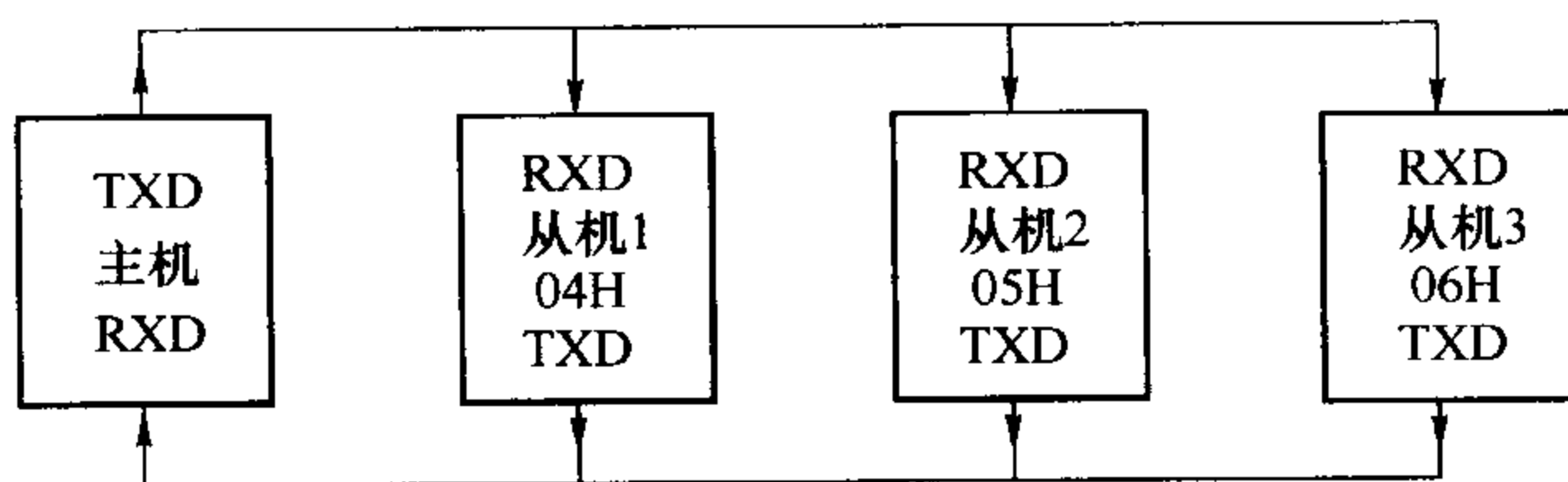


图 9.3.8 多机通信系统

在主机和某一从机系统通信之前,先将从机地址发送给各个从机系统,接着才传送数据。多机通信的过程如下。

① 所有从机的  $SM2=1$ ,处于只接收地址帧的状态。

② 主机发送一帧地址信息,其中包含 8 位地址,第 9 位为地址/数据标志位。第 9 位为 1,表示发送的是地址信息。

③ 各个从机接收到地址帧后,各自将所接收的地址与本从机的地址相比较。对于地址相符的从机,使本机的  $SM2=0$  以便接收主机随后发来的所有信息;对于地址不相符的从机,仍保持  $SM2=1$  状态,对主机随后发来的数据不予理睬,等待主机发送新的地址帧。

④ 主机发送控制指令与数据给被寻址的从机,数据帧的第 9 位置 0,表示发送的是数



据或控制指令。

图 9.3.8 所示的多机通信系统是主从式,由主机控制从机之间的通信,从机与从机之间的通信,只能经主机才能实现。

### 9.3.4 波特率的确定

在串行通信中,收发双方对发送和接收的数据速率有一定的约定,通过软件对单片机串行口编程可设定 4 种工作模式。其中,模式 0 和模式 2 的波特率是固定的;而模式 1 和模式 3 的波特率是可变的。

由于输入的移位时钟来源不同,因此各种模式的波特率计算公式也不相同。

#### 1. 模式 0 的波特率

根据模式 0 的工作原理,发送或接收一位数据的移位时钟由 S6P2(即第 6 个状态周期的第 2 个节拍)给出,即每个机器周期产生一个移位脉冲,发送或接收一位数据。因此,波特率固定为振荡器频率的 1/12,并不受 PCON 寄存器中 SMOD 位的影响。

$$\text{模式 0 的波特率} = f_{\text{osc}} / 12$$

#### 2. 模式 2 的波特率

串行口模式 2 波特率的产生与模式 0 不同,即输入的时钟源不同,其时钟输入部分如图 9.3.7 所示。

控制发送和接收的移位时钟由振荡器频率  $f_{\text{osc}}$  的第 2 节拍 P2 时钟(即  $f_{\text{osc}}/2$ )给出,所以,模式 2 的波特率与 PCON 中 SMOD 位的值有关:当 SMOD=0 时,波特率为  $f_{\text{osc}}$  的 1/64;当 SMOD=1 时,波特率为  $f_{\text{osc}}$  的 1/32。即:

$$\text{模式 2 的波特率} = \frac{2^{\text{SMOD}}}{64} \times f_{\text{osc}}$$

#### 3. 模式 1 和模式 3 的波特率

模式 1 和模式 3 的波特率由定时/计数器 1 或定时/计数器 2 的溢出速率来决定,通过 T2CON 中的 TCLK 和 RCLK 来选择。发送器的波特率由 TCLK 选择,TCLK=1 时发送波特率由定时/计数器 2 的溢出速率来决定,TCLK=0 时发送波特率由定时/计数器 1 的溢出速率来决定。接收器的波特率由 RCLK 选择,RCLK=1 时接收波特率由定时/计数器 2 的溢出速率来决定,RCLK=0 时接收波特率由定时/计数器 1 的溢出速率来决定。也可以用其中一个定时器确定发送波特率,用另一个确定接收波特率。

##### (1) 用 T1 产生波特率

当使用 T1 作为波特率发生器时,模式 1 和模式 3 的波特率就由 T1 的溢出速率和 SMOD 的值共同确定,其关系为:

$$\text{模式 1 和模式 3 的波特率} = 2^{\text{SMOD}} \times \text{T1 溢出速率} / 32$$

SMOD=0,波特率为 T1 溢出速率的 1/32;SMOD=1,波特率为 T1 溢出速率的 1/16。当定时器 T1 作为波特率发生器时,T1 可设置为定时器模式,也可以设置为计数器模式。T1 的溢出速率是和它的工作模式有关的,可编程设置为模式 0~2 中的任意一种。最典型的是将 T1 设置为 8 位自动重载的模式 2。此时,T1 中断应被禁止,波特率可由下式得出:

$$\text{模式 1 和模式 3 的波特率} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{\text{osc}}}{12 \times (256 - \text{TH1})}$$

如果将 T1 定义为 16 位定时器模式, 并利用 T1 的溢出中断控制重装常数, 就可以得到很低的波特率。表 9.3.1 列出了常用波特率值和对应的晶振频率, 以及在使用 T1 产生波特率时的工作模式和计数初值。

#### (2) 用 T2 产生波特率

用 T2 作波特率发生器时的结构原理图参见图 8.4.6。模式 1 和模式 3 的波特率与 T2 的关系为:

$$\text{模式 1 和模式 3 的波特率} = \text{T2 的溢出速率} / 16$$

由图 8.4.6 可以看出, T2 的溢出速率是由 T2 的工作模式确定的, T2 既可以工作在定时器模式, 也可以工作在计数器模式。最典型的应用是把 T2 设置为定时器, 且工作于 16 位的常数自动重装模式。16 位的常数值由软件装入 RCAP2H 和 RCAP2L 中。此时, T2 的计数输入脉冲来自系统振荡器时钟的二分频, 当 T2 计数溢出时, 溢出信号控制将 RCAP2H 和 RCAP2L 寄存器中的初值重新装入 TH2 和 TL2 中, 并从此值开始重新计数。由于 T2 的溢出速率是严格不变的, 因此使串行口模式 1 和模式 3 的波特率非常稳定, 其计算公式为:

$$\text{模式 1 和模式 3 的波特率} = \frac{f_{\text{osc}}}{2 \times 16 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

式中的 (RCAP2H, RCAP2L) 为两个寄存器内容组成的 16 位二进制数。表 9.3.2 列出了使用 T2 产生的常用波特率和对应的晶振频率以及所需要的计数初值。

表 9.3.2 由 T1 产生的常用波特率值

波特率 /bit · s <sup>-1</sup>	f <sub>osc</sub> /MHz	SMOD	T1			波特率 /bit · s <sup>-1</sup>	f <sub>osc</sub> /MHz	SMOD	T1		
			C/ $\bar{T}$	模式	重装值				C/ $\bar{T}$	模式	重装值
4 800	16	1	0	2	EFH	2 400	16	0	0	2	EFH
2 400	16	1	0	2	DDH	1 200	16	0	0	2	DDH
1 200	16	1	0	2	BBH	600	16	0	0	2	BBH
600	16	1	0	2	75H	300	16	0	0	2	75H
4 800	12	1	0	2	F3H	2 400	12	0	0	2	F3H
2 400	12	1	0	2	E6H	1 200	12	0	0	2	E6H
1 200	12	1	0	2	CCH	600	12	0	0	2	CCH
600	12	1	0	2	98H	300	12	0	0	2	98H
300	12	1	0	2	30H	110	12	0	0	1	FEEBH
56 800	11.059	1	0	2	FFH	9 600	11.059	0	0	2	FDH
19 200	11.059	1	0	2	FDH	4 800	11.059	0	0	2	FAH
9 600	11.059	1	0	2	FAH	2 400	11.059	0	0	2	F2H
4 800	11.059	1	0	2	F4H	1 200	11.059	0	0	2	E8H
2 400	11.059	1	0	2	E8H	600	11.059	0	0	2	D0H
1 200	11.059	1	0	2	D0H	300	11.059	0	0	2	A0H
600	11.059	1	0	2	A0H	1 200	6	0	0	2	F3H
300	11.059	1	0	2	40H	110	6	0	0	2	72H

由图 8.4.6 也可以看出,在计数状态( $C/\overline{T2}=1$ )的波特率为:

$$\text{模式 1 和模式 3 的波特率} = \frac{\text{外部时钟频率}}{16 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

在 T2 作为波特率发生器启动工作后,就不能通过软件对 TH2 和 TL2 进行读写,对 RCAP2H 和 RCAP2L 可以读但不可以写,否则会造成波特率错误。只有在 T2 停止计数后( $\text{TR2}=0$ ),才可以对 TH2,TL2 或 RCAP2H,RCAP2L 寄存器读写。表 9.3.3 列出了使用 T2 产生的常用波特率。

表 9.3.3 用 T2 产生的常用波特率

波特率 /bit · s <sup>-1</sup>	$f_{\text{osc}}/\text{MHz}$	RCAP2H	RCAP2L	波特率 /bit · s <sup>-1</sup>	$f_{\text{osc}}/\text{MHz}$	RCAP2H	RCAP2L
38 400	16	FFH	F3H	56 800	11.059	FFH	FAH
19 200	16	FFH	E6H	38 400	11.059	FFH	F7H
9 600	16	FFH	CCH	19 200	11.059	FFH	EEH
4 800	16	FFH	98H	9 600	11.059	FFH	DCH
2 400	16	FFH	30H	4 800	11.059	FFH	B8H
1 200	16	FEH	5FH	2 400	11.059	FFH	70H
600	16	FCH	BFH	1 200	11.059	FEH	E0H
300	16	F9H	7DH	600	11.059	FDH	C0H
110	16	EEH	3FH	300	11.059	FBH	80H
9 600	12	FFH	D9H	4 800	6	FFH	D9H
4 800	12	FFH	B2H	2 400	6	FFH	B2H
2 400	12	FFH	64H	1 200	6	FFH	64H
1 200	12	FEH	C8H	600	6	FEH	C8H
600	12	FDH	8FH	300	6	FDH	8FH
300	12	FBH	1EH	110	6	F9H	57H

## 9.4 串行通信应用举例

### 9.4.1 串行口模式 0 的应用

串行口工作在模式 0 时,是同步移位寄存器方式。单片机为了扩展输出口,常利用模式 0 的输出方式外接移位寄存器。

**例 9-1** AT89S52 的串行接口外接 74LS164 移位寄存器,每接一片 74LS164 可扩展一个 8 位并行输出口,用以连接一个 LED 作静态显示器或作键盘中 8 条列线使用。

图 9.4.1 为串行口扩展两位 LED 显示器的实用电路。

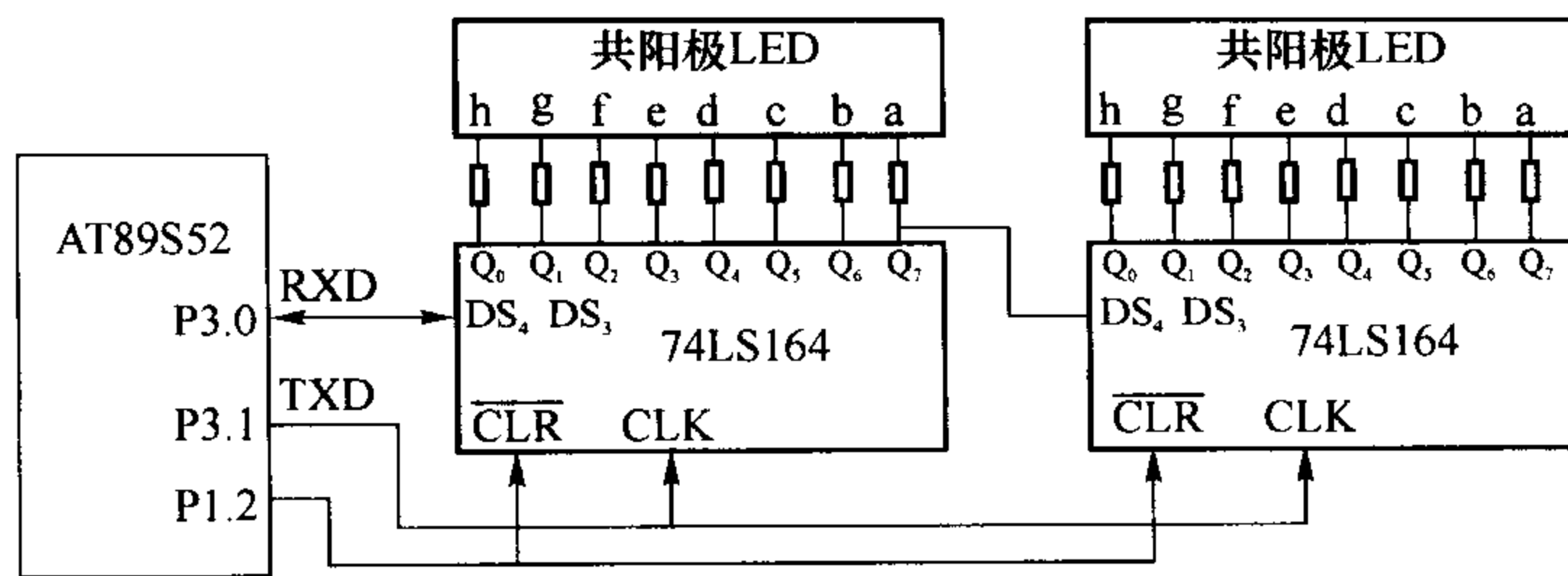


图 9.4.1 串行口扩展输出接口

### 解 (1) 功能分析

图 9.4.1 中,单片机的串行口工作在模式 0 时,RXD 用于输出数据,与 74LS164 的数据输入端相连,而 TXD 用于输出同步移位脉冲,接 74LS164 的 CLK 端,第一片 74LS164 的最高位接入第二片 74LS164 的最低位,移位过程用单片机的 P1.2 引脚控制。下面为从内部 RAM 的 61H 和 62H 单元中取出要显示的数据,查表获得 7 段显示码,由串行口送给显示器的程序清单,其中 R0 作为数据缓冲区指针。

### (2) 程序清单

```

                ORG      1100H                      ;子程序入口地址
DISP:          SETB     P1.2                        ;控制端打开
                MOV      R7, # 02H
                MOV      R0, # 61H
                MOV      SCON, # 00H                ;串行口方式 0 初始化
DL1:           MOV      A, @R0
                ADD      A, # 0EH
                MOVC     A, @A + PC                  ;查 7 段代码表
                MOV      SBUF, A                    ;发送数据
DL2:           JNB      TI, DL2                      ;等待发送完成
                CLR      TI
                INC      R0
                DJNZ     R7, DL1
                CLR      P1.2
LOOP:          AJMP     DONE
TAB:           DB 0C0H, 0F9H, 0A4H, 0B0H, 99H      ;7 段代码表
                DB 92H, 82H, 0F8H, 80H, 90H, 88H
                DB 83H, 0C6H, 0A2H, 86H, 84H

```

## 9.4.2 串行口模式 1 的应用

串行口工作在模式 1,属于波特率可变的 8 位通用异步收发器,其中波特率一般由定

时器的溢出速率来决定。

**例 9-2** 设有两个单片机应用系统相距很近,将它们的串行口直接相连,以实现全双工的双机通信。设甲机发送乙机接收,待发送的数据是标准的 ASCII 码,存储在内部 RAM 单元 20H~3FH 中,要求在最高位上加奇校验位后由串行口发送出去,发送的波特率为 1 200 bit/s,  $f_{osc}=11.059\text{ MHz}$ 。

**解** (1) 功能分析

7 位 ASCII 码加上 1 位奇偶校验位共 8 位数据,所以可以采用串行口模式 1 来完成。

单片机的奇偶校验位 P 是当累加器 A 中 1 的个数为奇数时,  $P=1$ 。如果直接把 P 的值放入 ASCII 码的最高位,恰好形成了偶校验,与要求不符。因此,要把 P 的值取反后放入 ASCII 码的最高位,才是要求的奇校验。

双工通信要求收、发能同时进行。实际上,串行口在采用中断方式工作的情况下,收、发主要是在串行口中进行,CPU 只负责把数据从接收缓冲器中读出和把数据写入发送缓冲器。在中断请求产生的情况下响应中断,通过检测是 RI 置位还是 TI 置位来决定 CPU 是进行发送操作还是进行接收操作。发送和接收都通过子程序来完成。

(2) 波特率的计算

串行口工作在模式 1,定时器 T1 工作在模式 2 作为波特率发生器。波特率计算公式为:

$$\text{波特率} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{osc}}{12} (256 - \text{TH1})$$

设  $\text{SMOD}=0$ ,则  $\text{TH1} = 256 - \frac{f_{osc}}{32 \times 12 \times 1\,200} = 256 - 24 = 232 = 0\text{E8H}$ ,当然也可以通过查表 9.3.2 得到时间常数 0E8H。

(3) 甲机发送程序

```

START:  MOV  TMOD, # 20H           ;T1 工作在模式 2
        MOV  TL1, # 0E8H          ;装载预置常数
        MOV  TH1, # 0E8H
        SETB TR1
        MOV  SCON, # 40H          ;串行口工作在模式 1
        MOV  R0, # 20H            ;数据区首地址
        MOV  R7, # 20H            ;数据块长度
LOOP:   MOV  A, @R0                ;取数据
        MOV  C, P                  ;取奇偶校验位
        CPL  C
        MOV  ACC. 7, C
        MOV  SBUF, A               ;启动发送
DONE:   JNB  TI, DONE
        CLR  TI                    ;软件清除 RI
        INC  R0
    
```

```

        DJNZ R7,LOOP
        AJMP START                ;循环发送
(4) 乙机接收程序
START:  MOV  TMOD,#20H            ;T1 工作在模式 2
        MOV  TL1,#0E8H          ;装载预置常数
        MOV  TH1,#0E8H
        SETB TR1
        MOV  SCON,#50H          ;串行口工作在模式 1
        MOV  R0,#20H            ;数据区首地址
        MOV  R7,#20H            ;数据块长度
DONE:   JNB  RI,DONE             ;等待接收结束
        CLR  RI
        MOV  A,SBUF             ;取数据
        MOV  C,P                ;检查奇偶校验位
        CPL  C
        ANL  A,#7FH            ;去掉校验位
        JC   ERROR             ;转去出错处理
        MOV  @R0,A              ;保存所接收数据
        INC  R0
        DJNZ R7,DONE
ERROR:  ...                     ;出错处理程序

```

### 9.4.3 串行口模式 2 和模式 3 的应用

串行口的模式 2 与模式 3 基本一样,只是波特率设置不同,都是接收和发送 11 位信息:开始为 1 位起始位(0),中间是 8 位数据位,数据位之后是 1 位可编程控制位,最后是 1 位停止位。与方式 1 相比,只是多了 1 位可编程控制位。

**例 9-3** 编写串行发送程序,被发送的数据存储在内部 RAM 的 30H~4FH 单元中,要求每个数据要加上奇偶检验。

**解** (1) 功能分析

RAM 的 30H~3FH 单元中存储着普通的 8 位数据,要求把每个数据的奇偶校验位一同发送出去,则每帧数据的基本信息为 9 位,所以可以采用模式 2 或者模式 3 来进行发送。本例中,串行口设定为模式 2,9 位波特率固定的 UART, TB8 作奇偶校验位。在数据写入发送缓冲器之前,先将数据的奇偶校验位 P 写入 TB8,在采用模式 2 的情况下,单片机会在发送完基本的 8 位数据后,自动把 TB8 的数据也发送出去,接收端把接收到的第 9 位数据存储在自己的 TB8 中。本例采用查询和中断两种方式发送。

(2) 采用查询方式的程序清单

```

        ORG  0000H
        AJMP START

```

```

                ORG 0100H
START:  MOV  SCON, #80H           ; 串行口工作在模式 2
        MOV  PCON, #80H         ; 取波特率为  $f_{osc}/32$ 
        MOV  R0, #30H           ; 数据段首址
        MOV  R7, #20H           ; 数据段长度
LOOP:   MOV  A, @R0
        MOV  C, P
        MOV  TB8, C             ; 奇偶校验标志送入 TB8
        MOV  SBUF, A            ; 数据发送
WAIT:   JBC  TI, NEXT           ; 查询是否一帧数据发送完毕
        SJMP WAIT
NEXT:   INC  R0
        DJNZ R7, LOOP           ; 未完, 则发送下一个数据
HERE:   AJMP HERE

```

### (3) 采用中断方式的程序清单

```

                ORG 0000H
                AJMP START
                ORG 0023H        ; 串行口中断入口地址
                AJMP SINT        ; 转到串行口中断服务程序
                ORG 0100H
START:  MOV  SP, #60H           ; 设置堆栈指针
        MOV  SCON, #80H        ; 串行口工作在模式 2
        MOV  PCON, #80H        ; 取波特率为  $f_{osc}/32$ 
        MOV  R0, #30H
        MOV  R7, #20H
        SETB ES                 ; 串行口开中断
        SETB EA                 ; 系统开中断
        MOV  A, @R0
        MOV  C, P
        MOV  TB8, C             ; 奇偶校验标志送入 TB8
        MOV  SBUF, A            ; 数据发送
HERE:   AJMP HERE              ; 等待中断
SINT:   CLR  TI
        INC  R0
        MOV  A, @R0
        MOV  C, P
        MOV  TB8, C
        MOV  SBUF, A            ; 数据发送

```



```

        DJNZ R7,ENDS          ;未完,则转去中断返回
        CLR  ES               ;所有数据发完,禁止串行口中断
ENDS:   RETI                  ;中断返回

```

**例 9-4** 编写串行接收程序,接收 10H 字节的数据存储在内部 RAM 的 30H~4FH 单元中。设串行口工作于模式 3,接收到的第 9 位数据为奇偶校验位,波特率为 2 400 bit/s,  $f_{osc}=11.059\text{ MHz}$ 。

**解** (1) 功能分析

模式 3 为波特率可变的 9 位 UART,波特率取决于定时器的溢出速率。对于波特率的计算,可以采用查表的方法,也可以利用公式直接计算。本例中选择定时/计数器 T2 作为串行口波特率发生器。

(2) 波特率计算

串行口工作在模式 3,定时器 T2 定义为波特率发生器模式。波特率计算公式为:

$$\text{波特率} = \frac{f_{osc}}{2 \times 16 \times [65\,536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

则

$$\text{RCAP2H}, \text{RCAP2L} = 65\,536 - \frac{f_{osc}}{2 \times 16 \times 2\,400} = 65\,392 = 0\text{FF}70\text{H}$$

(3) 源程序清单

```

MAIN:   MOV    R0, # 30H          ;接收数据的存储区
        MOV    R7, # 20H
        ACALL  INITS              ;调用初始化串行口子程序
WAIT:   JBC    RI, CHECK          ;等待接收数据结束
        SJMP   WAIT
CHECK:  MOV    A, SBUF            ;以下为检查所接收数据的正确性
        JNB    P, PNP             ;P = 0, 转 PNP
        JNB    RB8, ERROR         ;P = 1, RB8 = 0, 转出错处理
        SJMP   RIGHT
PNP:    JB     RB8, ERROR          ;P = 0, RB8 = 1, 转出错处理
RIGHT:  MOV    @R0, A             ;数据存储
        INC    R0
        DJNZ   R7, WAIT           ;是否接收下一帧数据
        CLR    PSW. 5             ;正确接收完毕标志
        RET
ERROR:  SETB   PSW. 5             ;非正确接收标志
        RET
INITS:  MOV    A, # 0FFH          ;装定时常数
        MOV    RCAP2H, A
        MOV    TH2, A

```

```
MOV    A, #70H
MOV    RCAP2L, A
MOV    TL2, A
MOV    T2CON, #34H      ;使用 T2 作波特率发生器
MOV    SCON, #0D0H
RET
```

## 习 题

1. 串行通信和并行通信各有什么优缺点？它们分别适用于什么场合？
2. 异步通信和同步通信的工作原理是什么？
3. 串行口有几种工作方式？它们各有什么特点？
4. 单片机中 SCON 的 SM2, TB8 和 RB8 有什么作用？
5. 简述串行口接收和发送数据的过程。
6. 某异步通信接口,其帧格式由 1 位起始位 0, 7 位数据位, 1 位奇偶校验位和 1 位停止位 1 组成。当该接口每分钟发送 1 800 个字符时,请计算出波特率。
7. 若晶振频率为 11.059 MHz, 串行口工作在方式 1, 波特率为 4 800 bit/s。分别写出用 T1 和 T2 作为波特率发生器的方式字和计数初值。
8. 若定时器 T1 设置成模式 2 作波特率发生器, 已知频率为 6 MHz。求可能产生的最高波特率和最低波特率。
9. 试设计一个单片机的双机通信系统, 并编写通信服务程序, 将甲机内部 RAM 30H~3FH 存储区的数据块通过串行口传送到乙机内部 RAM 中的 40H~4FH 存储区中去。
10. 串行口工作在模式 1 和模式 3, 其波特率与频率、定时器 T1 工作模式 2 的初值及 SMOD 位的关系如何？已知频率为 6 MHz, 现分别利用 T1 和 T2 产生 110 bit/s 的波特率, 试计算定时器初值。

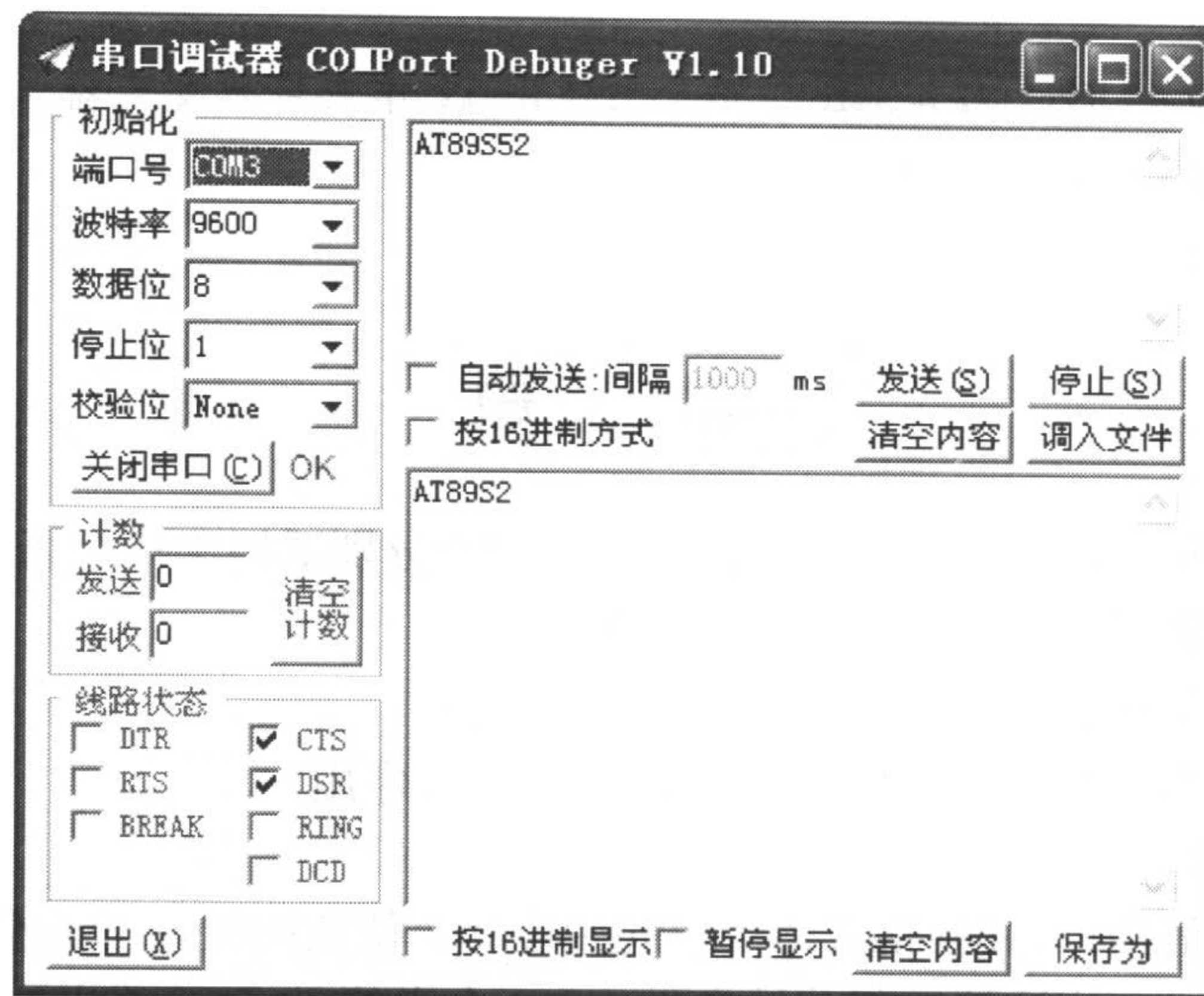
## 实 践 训 练

学完本章内容后, 为掌握串行口的相关知识、了解串口的的基本使用方法、掌握串口程序的编程方法, 可按照下述步骤进行实践训练。

按照图 9.2.4 电路, 将单片机与计算机相连接, 进行串口通信训练, 计算机的接收、发送程序利用 comdebug.exe 或串口大师等软件进行调试。COMDEBUG 界面如下图所示, 按照界面提示进行端口号、波特率、数据位、停止位、校验位的设置, 并与单片机的设置相同。如发送和接收的是十六进制数据, 则选中“按十六进制方式”, 若是 ASCII 码, 则不用选中该框。为了检测发送、接收状态, 单片机可设置两个发光管进行状态显示, 一个显示发送状态, 一个显示接收状态。进行以下调试:

- (1) 改变不同的波特率 1 200 bit/s, 2 400 bit/s, 4 800 bit/s, 9 600 bit/s;
- (2) 利用定时器 1 作波特率发生器, 并使其工作在不同模式;

- (3) T2 作波特率发生器;
- (4) 加奇偶校验。



# 第 10 章 单片机应用系统扩展技术

对于简单应用场合,单片机最小系统基本能够满足功能要求。但对于复杂的应用场合,可利用一些专用集成电路对单片机进行系统扩展,构成功能更强、规模较大的应用系统。系统扩展一般包括总线、程序存储器、数据存储器、I/O 接口、EEPROM、A/D、D/A 等的扩展。具体扩展哪一部分,需视具体情况而定。但不管扩展哪一部分,都要尤其注意地址总线、数据总线、控制总线的连接。

## 10.1 总线扩展及地址分配

在介绍一些常用的外围接口芯片之前,应先了解总线扩展方法和相应地址分配原则,这将为设计单片机硬件电路和软件编程奠定基础。

### 10.1.1 总线扩展

AT89S52 单片机有很强的外部扩展能力,大部分常规集成电路芯片可用于单片机的扩展电路。扩展的内容主要有总线、程序存储器、数据存储器、I/O 口扩展、A/D 和 D/A 扩展、中断扩展等。但由于受引脚的限制,AT89S52 单片机 P0 口是分时复用的地址/数据总线,而且与 I/O 口线复用,为了将地址总线与数据总线分离出来,以便同片外的电路正确连接,需要在单片机外部增加地址锁存器,构成片外三总线结构,即地址总线、数据总线和控制总线结构,如图 10.1.1 所示。

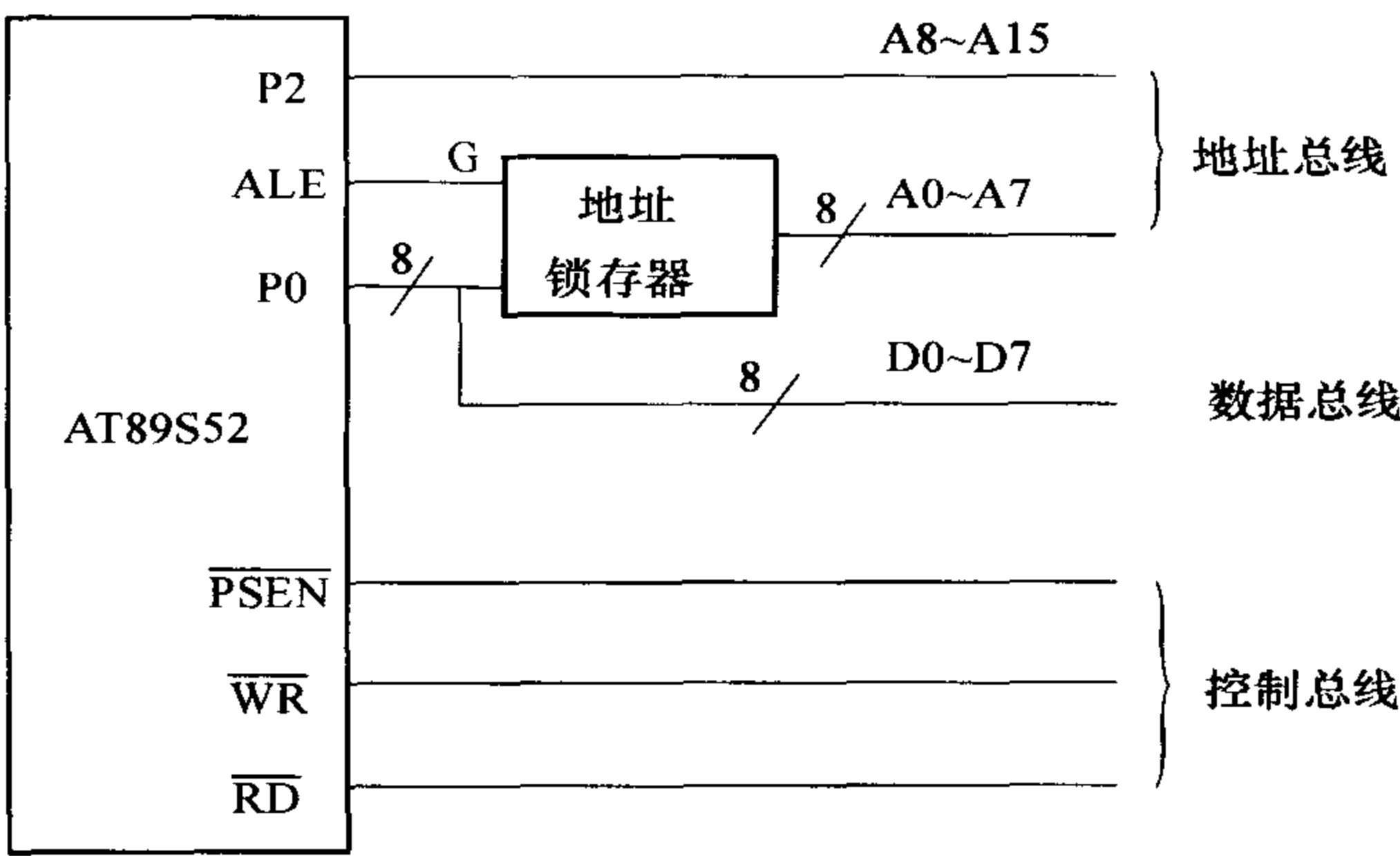


图 10.1.1 AT89S52 扩展的三总线

#### 1. 地址锁存器

地址锁存器可使用带三态缓冲输出的 8-D 锁存器 74LS373,8282 或带清除端的 8-D

锁存器 74LS273。74LS373 地址锁存器的引脚配置及结构如图 10.1.2 所示。

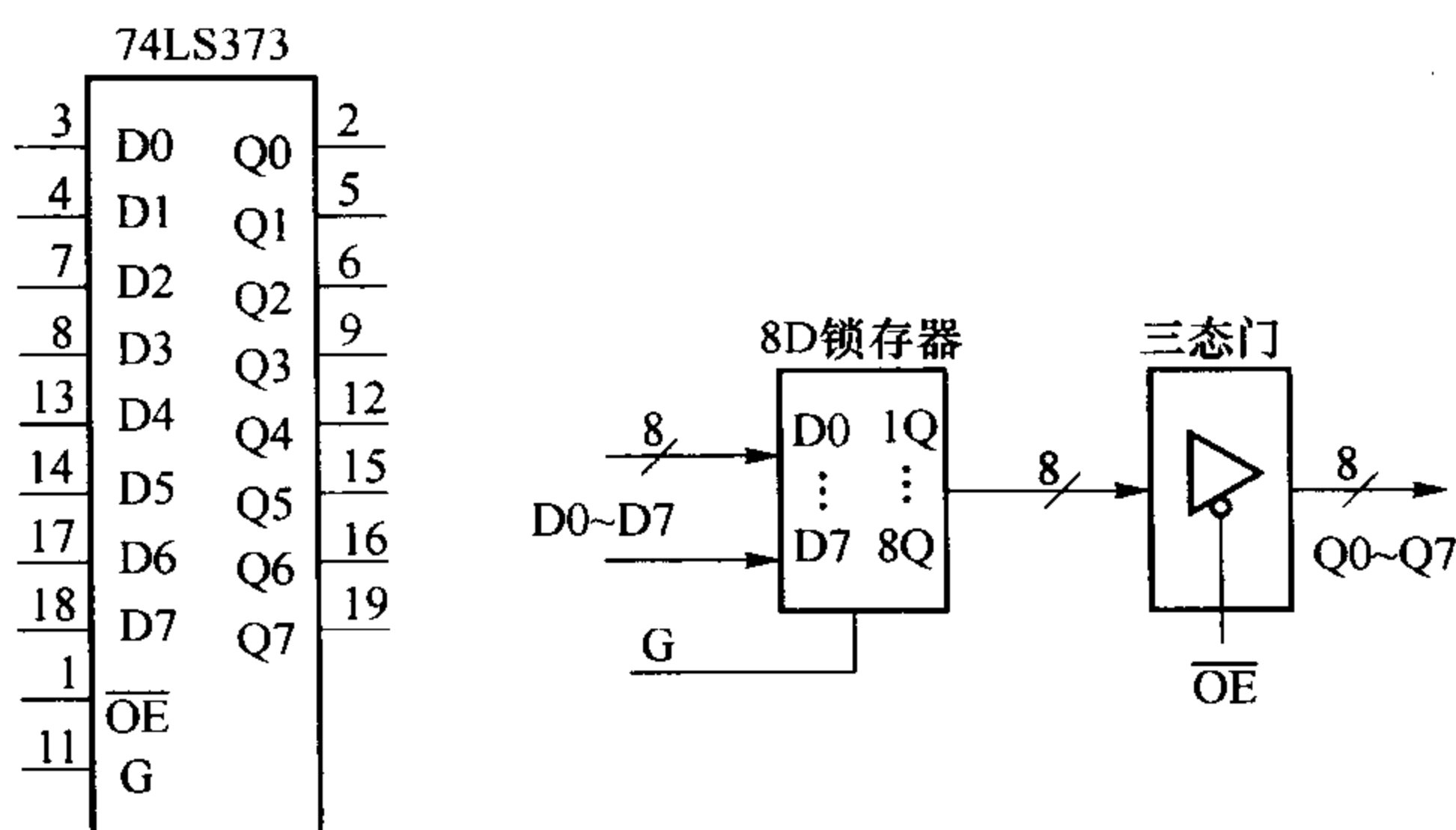


图 10.1.2 74LS373 地址锁存器引脚图及结构图

74LS373 有直通、高阻和锁存 3 个状态。通过锁存信号输入端  $G$  和输出允许控制信号输入端  $\overline{OE}$  组合,可实现上述 3 个状态。

当三态门的  $\overline{OE}$  为低电平且  $G=1$  时,三态门处于直通状态,允许  $1Q \sim 8Q$  输出到  $Q0 \sim Q7$ ;当  $\overline{OE}$  为高电平时,输出三态门断开,输出线  $Q0 \sim Q7$  处于高阻状态;当  $\overline{OE}=0$  且  $G$  端出现下降沿时,为锁存状态。

74LS373 作为地址锁存器时,首先应使三态门的使能信号  $\overline{OE}$  为低电平,这时,当  $G$  输入为高电平时,锁存器直通状态,此时输出端  $Q0 \sim Q7$  状态和输入端  $D0 \sim D7$  状态相同;当  $G$  端从高电平到低电平(下降沿)时,输入端  $1D \sim 8D$  的数据锁入  $1Q \sim 8Q$  的 8 位锁存器,为锁存状态。

74LS373 作为地址锁存器时,它的锁存控制端  $G$  直接与单片机的锁存控制信号  $ALE$  相连, $ALE$  下降沿进行低 8 位地址锁存,直到下一次  $ALE$  变高时,地址才发生变化。

## 2. 三总线

AT89S52 单片机的片外引脚可构成如图 10.1.3 所示的三总线结构,通过扩展的三总线,单片机可以方便地扩展其外部数据存储器,外部程序存储器及 I/O 接口,同时外围芯片也可以通过这三总线进行扩展。

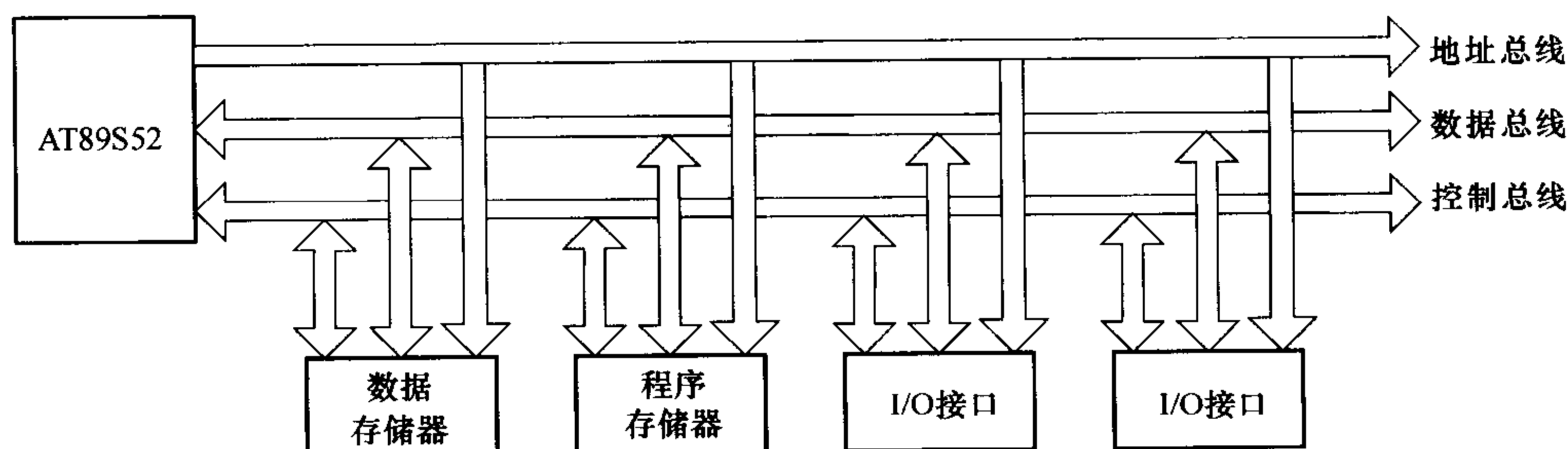


图 10.1.3 AT89S52 系统扩展及接口结构

### (1) 地址总线(AB)

地址总线由单片机 P0 口提供低 8 位地址  $A0 \sim A7$ ,P2 口提供高 8 位地址  $A8 \sim A15$ 。

如果 P2 口的全部 8 位口线作为高 8 位地址再加上 P0 口的 8 位口线作为低 8 位地址,可以形成 16 位地址,使单片机系统的寻址范围达到 64 KB 的最大范围。

P0 口是地址总线低 8 位和 8 位数据总线复用口,只能分时作为地址线。故 P0 口输出的低 8 位地址 A0~A7 必须用锁存器锁存。

### (2) 数据总线(DB)

数据总线由 P0 口提供,用 D0~D7 表示。P0 口为真正双向口,是应用系统中使用最为频繁的通道。单片机与外部交换的大部分数据、指令、信息都通过 P0 口传送。

### (3) 控制总线(CB)

系统扩展时还需要一些控制信号来构成控制总线,其中一部分是第一功能信号口线,另一部分是第二功能信号口线。包括:

- ① ALE,地址锁存选通信号,实现低 8 位地址的锁存;
- ②  $\overline{\text{PSEN}}$ ,扩展程序存储器读选通信号;
- ③  $\overline{\text{EA}}$ ,内、外程序存储器选择信号;
- ④  $\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ ,扩展数据存储器 and I/O 接口的读、写选通信号。

可以发现,经过上述系统扩展以后,剩下的只有 P1 口和 P3 的部分口线可以作为一般 I/O 接口使用。

数据总线并联到多个连接的外围芯片的数据线上,而在同一时间里只能有一个是有效的数据传送通道。哪个芯片的数据通道有效,则由地址线控制各个芯片的片选线来选择。

## 10.1.2 地址分配

AT89S52 单片机的地址总线有 16 位,P0 口经过锁存器输出低 8 位地址,P2 口输出高 8 位地址。所谓地址分配指利用单片机的地址总线(P2 口和 P0 口),通过适当的连接,最终达到一个地址唯一对应一个外围芯片的一个选中单元的目的,保证单片机的数据总线分时地与不同地址的外围芯片进行数据传送而不发生冲突。

首先,AT89S52 单片机的程序存储器与数据存储器使用不同控制信号和指令进行读/写操作,它们的地址可以重叠使用,不会因为地址重叠产生数据冲突问题。

其次,由于外部数据存储器和 I/O 接口是统一编址的,因此用户可以把外部 64 KB 的数据存储器空间的一部分作为扩展外围 I/O 的地址空间。这样,单片机就可以像访问外部 RAM 存储器那样利用 MOVX 指令访问外部接口芯片,对其进行读入或写出操作。

AT89S52 通过地址总线输出地址,可以选择某一外部存储器单元并对其进行读入或写出操作。要保证正确完成这种功能,需要经过两种选择:一是必须选择该存储器芯片或 I/O 接口芯片,称为片选;二是必须选择该芯片的某一存储单元,称为字选。高位片选地址加上字选单元地址,构成外围芯片的唯一地址。

常用的对外围芯片的片选方法分两种:线选择法(线选法)和地址译码选通法(译码法)。

## 1. 线选法

所谓线选法通常是把 P2 口的某一口线接到扩展外围芯片的片选端(一般是 $\overline{CS}$ ),当该口线为低电平时,就选中该芯片。

图 10.1.4 中包括 I/O 扩展芯片 8155 和 8255,2 KB 的数据存储器 6116、D/A 变换器 DAC0832(这些外围芯片将在本章予以详细介绍)。外围芯片的地址由片选地址和片内字选地址组成,而片内字选地址是由低位地址线进行全译码选择。决定该外围芯片的地址时,对于未用到的地址位一般均设成 1 状态,将它们推向高位,也可设成 0 状态。这样在得到的 16 位地址码中,既包括了片选控制,也包含了字选控制。根据图 10.1.4 中片选线的连接方法,地址译码如表 10.1.1 所示。

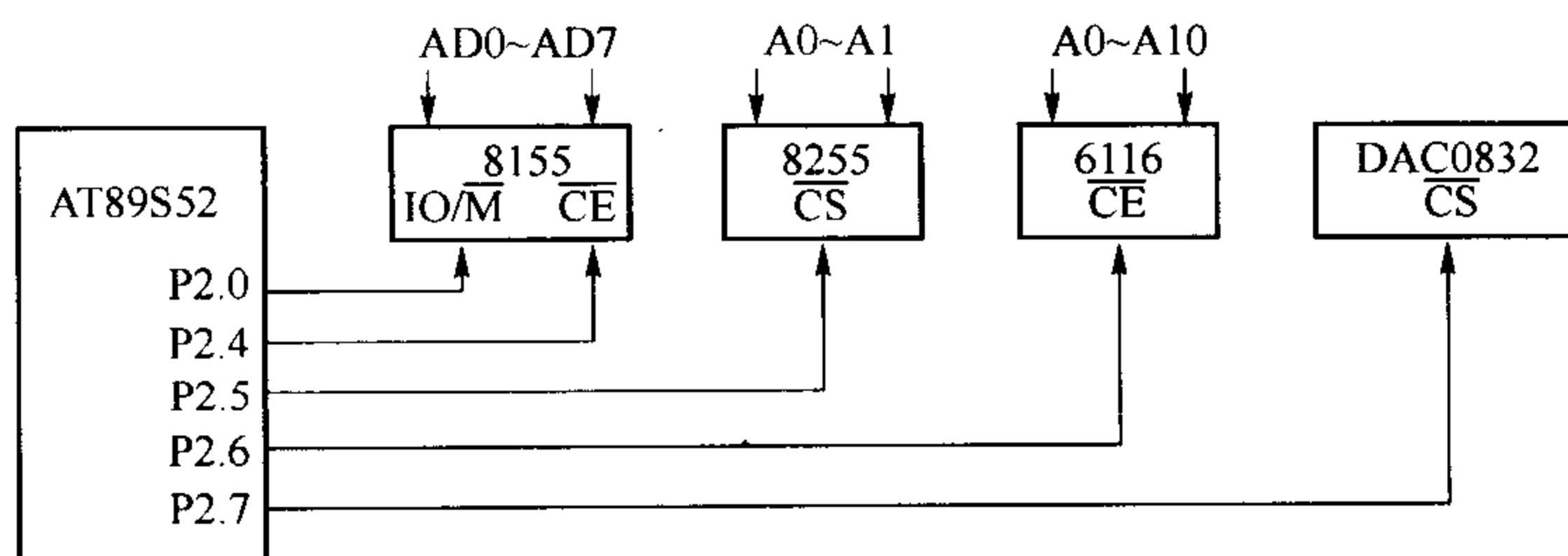


图 10.1.4 线选法地址译码

表 10.1.1 图 10.1.4 线选法地址表

外围器件		片内地址单元	地址选择线(A15~A0)	地址编码
8155	RAM	256 B	1 1 1 0 1 1 1 0 × × × × × × × ×	EE00H~EEFFH
	I/O	6 B	1 1 1 0 1 1 1 1 1 1 1 1 1 1 × × ×	EFF8H~EFFFH
8255		4 B	1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 × ×	DFFCH~DFFFH
6116		2 KB	1 0 1 1 1 × × × × × × × × × × × ×	B800H~BFFFH
DAC0832		1 B	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	7FFFH

例如访问外部数据存储器 6116 时,单片机通过地址总线选通 6116 数据存储器,通过数据总线对 6116 进行读/写操作。如执行:MOV DPTR, # B868H, MOVX A, @DPTR 程序执行时,P2 口出现 B8H,P0 口 68H,同时 $\overline{RD}=0$ ,使 6116 的 B868H 单元的数据出现在 P0 口,即读入了 B868H 单元的内容。

线选法的缺点是地址空间没有充分利用,各扩展芯片之间的地址范围不连续。

## 2. 地址译码选通法

所谓地址译码选通法通常是取扩展外围电路中最大容量芯片的地址线位数,作为芯片的字选,用于确定片内地址,用译码器对剩余的高位地址线进行译码,译出的信号作为片选线。片选线连接到扩展外围芯片的片选端上,当该口线为低电平时,就选中该芯片。

除了参加字选的地址线外,剩余的高位地址线全部用于译码的方式,称为全地址译码。图 10.1.5 所示为采用 74LS138 作为地址译码器的全地址译码电路,6264 字选需要



的地址线为 13 条,则剩余的 3 根地址线可作为片选线。译码器的 8 根输出线分别对应于一个 8 KB 的地址空间。

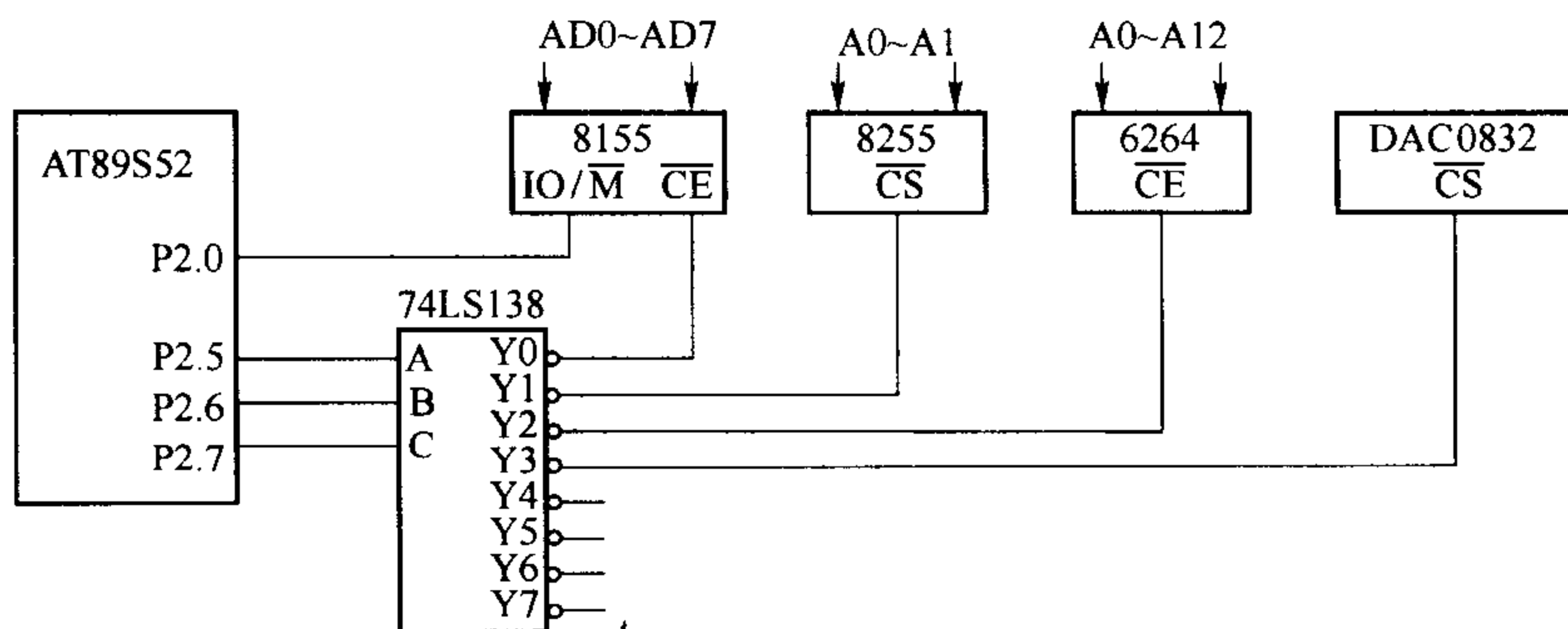


图 10.1.5 全地址译码

根据图 10.1.5 中的片选线的连接方法,全地址译码如表 10.1.2 所示。

表 10.1.2 图 10.1.5 全地址译码

外围器件	片内地址单元	地址选择线(A15~A0)	地址编码
8155	RAM	256 B	0 0 0 1 1 1 1 0 × × × × × × × ×
	I/O	6 B	0 0 0 1 1 1 1 1 1 1 1 1 1 1 × × ×
8255	4 B	0 0 1 1 1 1 1 1 1 1 1 1 1 1 × ×	3FFCH~3FFFH
6264	8 KB	0 1 0 × × × × × × × × × × × × × ×	4000H~5FFFH
DAC0832	1 B	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	7FFFH

采用地址译码法,可将原地址空间划分成符合需要的合理的若干区域,被划分的各块之间的地址连续。

## 10.2 外部程序存储器扩展

AT89S52 单片机的程序存储器空间和数据存储器空间相互独立。程序存储器和数据存储器空间最大可分别扩展至 64 KB。由于 AT89S52 的数据存储器和 I/O 的地址空间是统一编址的,在 64 KB 的外部 RAM 空间中,可划出一定的区间作为外部扩展接口的地址空间。

### 10.2.1 常用 EPROM 芯片

EPROM 是可擦除、可编程只读存储器,由独立的编程器进行编程。在重新改写程序时,通常要把 EPROM 芯片从系统中取下来,放到紫外线擦除器中照射擦除,然后才能重写。EPROM 典型产品有 2764,27128,27256,27512 等,这些产品均可用于 AT89S52 的程序存储器的系统扩展中。它们的引脚配置基本相同,只是在容量不同时地址口线不同。下面以 2764 和 27128A 为例,介绍它的引脚及性能。

2764A 是一种  $8\text{ K} \times 8$  位的紫外线擦除电可编程的只读存储器, 单一 +5 V 电源供电, 工作电流为 75 mA, 维持电流为 35 mA, 读出时间最长为 250 ns。28 脚双列直插式封装的引脚配置和引脚含义如图 10.2.1 所示。

27128A 是  $16\text{ K} \times 8$  位紫外线擦除电可编程的只读存储器, 单一 +5 V 电源供电, 工作电流为 100 mA, 维持电流为 40 mA, 读出时间最长为 250 ns。28 脚双列直插式封装的引脚配置和含义如图 10.2.1 所示。A12~A0 为地址线; 其余引脚与 2764A 相同。

EPROM 一般有以下 5 种工作方式。

(1) 读方式: 进入这种方式的条件是使片选控制线  $\overline{\text{CE}}$  为低, 同时让输出允许控制线  $\overline{\text{OE}}$  为低, 就可将指定单元的内容从数据总线上读出。

(2) 维持方式: 当片选控制线为高电平时, 芯片进入维持方式, 这时输出数据线呈高阻悬浮状态, 不占用系统数据总线。

(3) 编程方式: 在  $V_{\text{PP}}$  端加上规定的电压,  $\overline{\text{CE}}$  和  $\overline{\text{OE}}$  端加上合适的电平, 就能将数据线上的数据固化到指定的地址单元。

(4) 编程校验方式: 在  $V_{\text{PP}}$  端保持相应的电压, 再按读出方式操作, 读出编程固化好的内容, 以便校验写入的内容是否正确。

(5) 编程禁止方式: 当片选信号  $\overline{\text{CE}}$  无效时, 输出呈高阻状态。

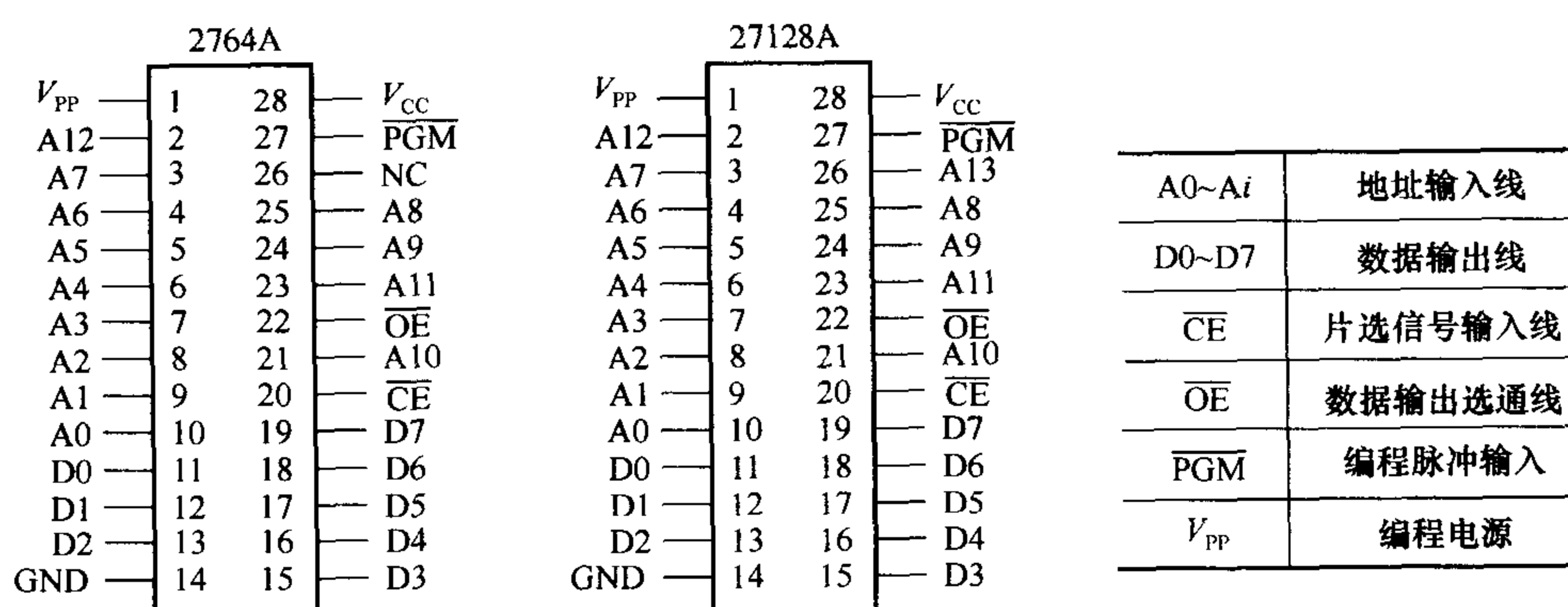


图 10.2.1 2764A 和 27128A 芯片引脚图

各种工作方式的选择通过在不同的控制线上加不同的电压来实现。

2764A 和 27128A 工作方式与控制线所加电压的关系, 如表 10.2.1 所示。

表 10.2.1 2764A 和 27128A 工作方式选择

引 脚 方 式	$\overline{\text{CE}}(20)$	$\overline{\text{OE}}(22)$	$\overline{\text{PGM}}(27)$	$V_{\text{PP}}/V(1)$	$V_{\text{CC}}/V(28)$	输出 (11~13, 15~19)
读	$V_{\text{IL}}$	$V_{\text{IL}}$	$V_{\text{IH}}$	5	5	Dout
维持	$V_{\text{IH}}$	任意	任意	5	5	高阻
编程	$V_{\text{IL}}$	$V_{\text{IH}}$	$V_{\text{IL}}$	12.5	5	Din
编程校验	$V_{\text{IL}}$	$V_{\text{IL}}$	$V_{\text{IH}}$	12.5	5	Dout
编程禁止	$V_{\text{IH}}$	任意	任意	12.5	5	高阻

### 10.2.2 典型 EPROM 扩展电路实现

随着大规模集成电路技术的发展,大容量存储器芯片的产量剧增,售价不断降低,其性能价格比明显增高。所以,在设计单片机系统时,应优先采用大容量存储芯片。这样,不仅可以使电路板的体积减小,成本降低,还可以降低系统功耗和减少控制逻辑电路,从而提高系统的稳定性和可靠性。

AT89S52 外扩 16 KB EPROM 27128 的电路图如图 10.2.2 所示。图中由 AT89S52、74LS373 和 27128 构成单片机最小系统。74LS373 的三态控制端  $\overline{OE}$  接地,以保持输出直通;G 端与 AT89S52 的 ALE 连接,每当 ALE 端的电平出现下降沿时,74LS373 锁存低 8 位地址 A7~A0,并输出给 27128 使用。

27128 是 16 K×8 位的 EPROM 芯片,用于存放程序和常数。它有 14 条地址线 A13~A0,可选择  $2^{14}=16\,384$  个存储单元。这 14 条地址线分别与 AT89S52 的 P0 口和 P2.0~P2.5 连接,当 AT89S52 发送 14 位地址信息时,可分别选中 27128 片内 16 KB 存储器中任何一个单元。27128 的  $\overline{CE}$  接地表示选中该芯片。27128 的  $\overline{OE}$  端由 AT89S52 的  $\overline{PSEN}$  引脚信号控制,当  $\overline{PSEN}$  信号出现下降沿时,允许 27128 输出,所指定的 27128 存储单元的内容送到 P0 口,在  $\overline{PSEN}$  的上升沿将数据送入单片机 CPU 内。

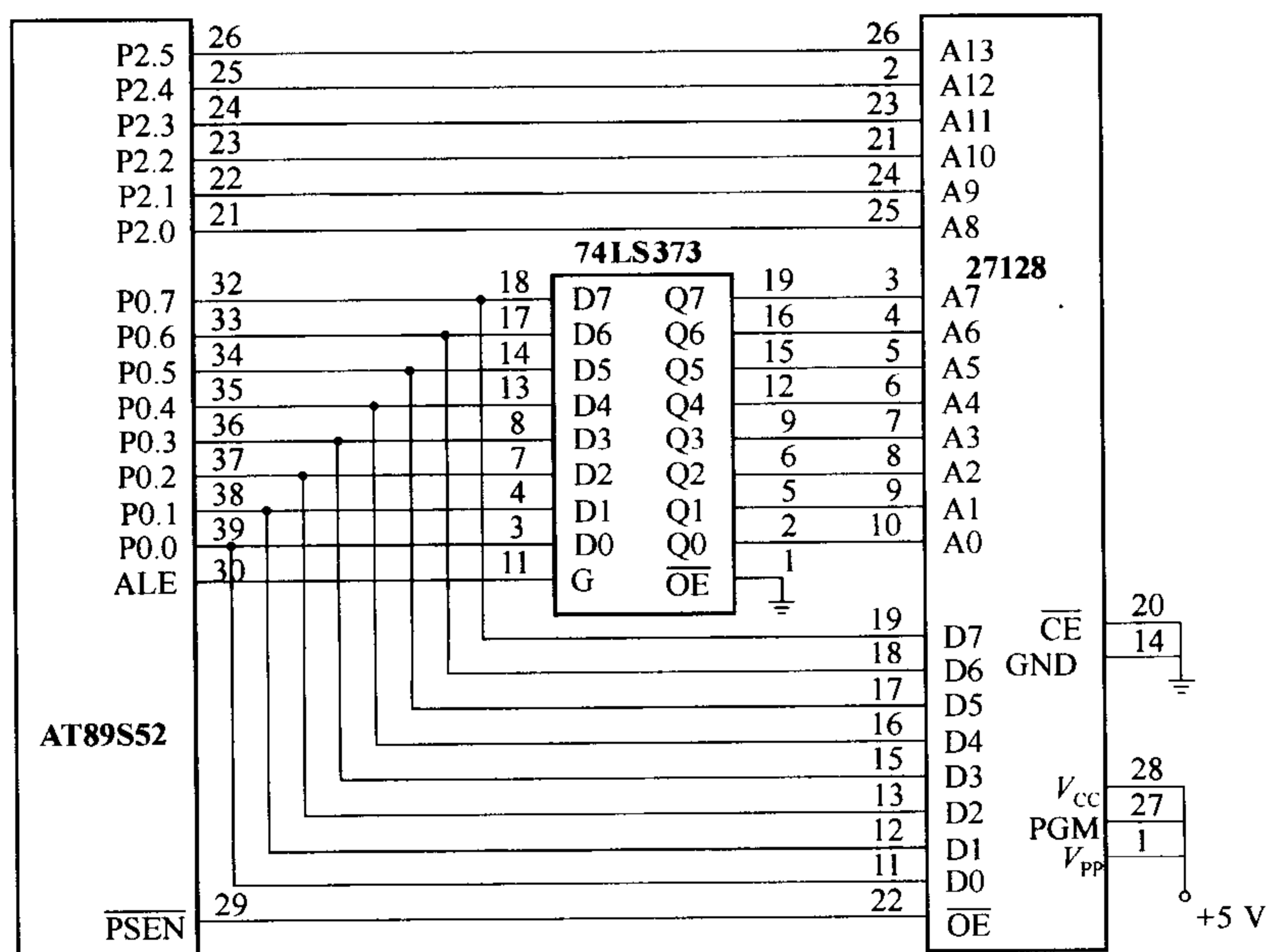


图 10.2.2 AT89S52 与 27128 的连线图

当 AT89S52 的外部程序存储器地址允许输入端  $\overline{EA}$  接高电平时, CPU 只访问片内 8 KB 程序存储器中的指令,但当程序计数器的值超过 8 KB 时,将自动转去执行片外 27128 程序存储器内的程序;当  $\overline{EA}$  引脚接低电平时, CPU 只访问外部 27128 程序存储器并执行外部程序存储器中的指令,而不访问片内程序存储器。

访问 ROM/EPROM 的指令为 `MOVC A, @A+PC` 或 `MOVC A, @A+DPTR`。

如读取 EPROM 地址为 1000H 单元内容的程序为：

```
MOV    DPTR, #1000H
```

```
MOVC   A, @A + DPTR
```

## 10.3 外部数据存储器的扩展

AT89S52 单片机内部有 256 B RAM 存储器。CPU 对内部 RAM 具有丰富的操作指令；但在用于实时数据采集和处理时，仅靠片内提供的 256 B 数据存储器远远不够，此时可扩展外部数据存储器。常用的数据存储器有静态 RAM 和动态 RAM 两种，与动态 RAM 相比，静态 RAM 无须考虑为保持数据而设置的刷新电路，故扩展电路较简单；但由于静态 RAM 是通过有源电路来保持存储器中的数据，因此要消耗较多的功率，价格也较高。EEPROM 芯片也可作为外部数据存储器，而且掉电后信息不丢失。本节主要讨论静态 RAM、并行 EEPROM、串行 EEPROM 与 AT89S52 的接口。

### 10.3.1 RAM(SRAM)的扩展

#### 1. 静态 RAM 的引脚及工作方式

常用的静态 RAM 芯片有 6116, 6264, 62128, 62256 等。引脚排列如图 10.3.1 所示。

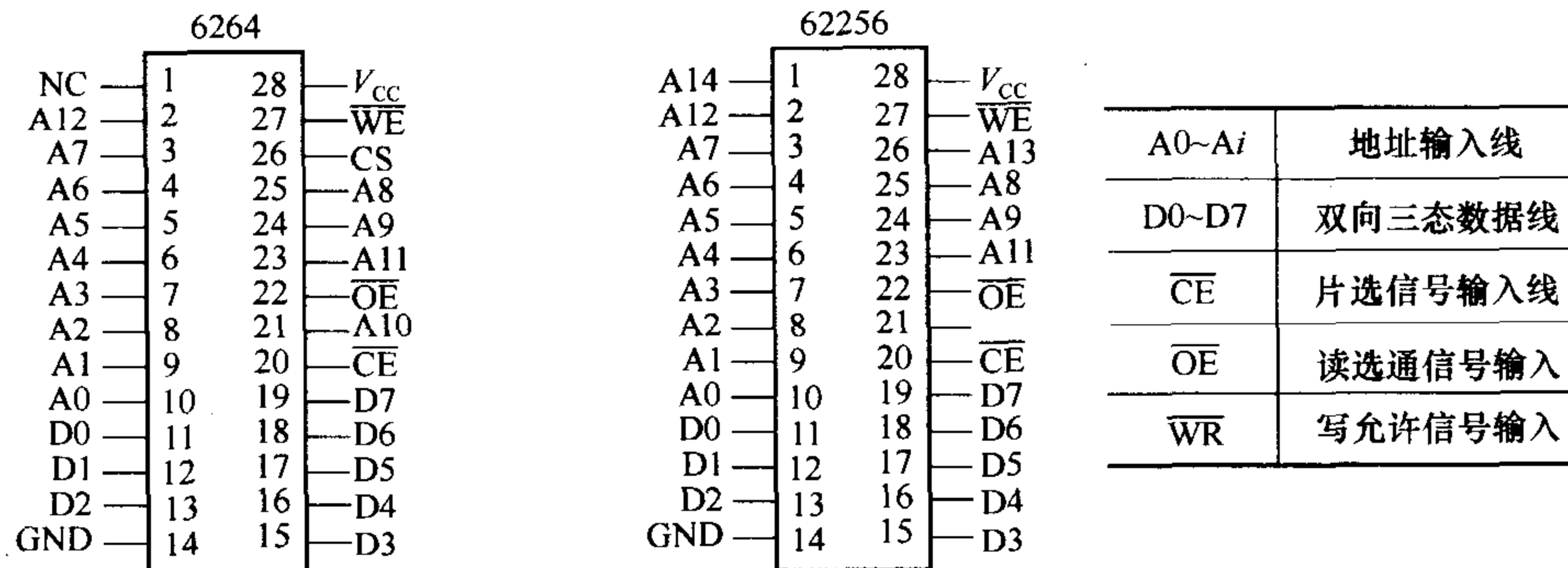


图 10.3.1 常用静态 RAM 芯片引脚图

静态 RAM 有读出、写入和维持 3 种工作方式，工作方式选择如表 10.3.1 所示。

表 10.3.1 静态 RAM 的工作方式选择

引 脚 方 式	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	输出(D0~D7)
读	$V_{IL}$	$V_{IL}$	$V_{IH}$	数据输出
写	$V_{IL}$	$V_{IH}$	$V_{IL}$	数据输入
维持	$V_{IH}$	任意	任意	高阻态

## 2. 外部静态数据存储器的扩展电路

静态数据存储器与单片机连接时,主要解决地址线、数据线和控制线的连接。P2 口提供高 8 位地址,P0 口分时提供低 8 位地址和 8 位双向数据总线。外部 SRAM 的读/写信号 $\overline{\text{OE}}$ 和 $\overline{\text{WR}}$ 分别由 AT89S52 的 $\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ 信号控制。片选端 $\overline{\text{CE}}$ 由地址译码器输出控制。图 10.3.2 给出了用线选法扩展 AT89S52 外部数据存储器的电路。

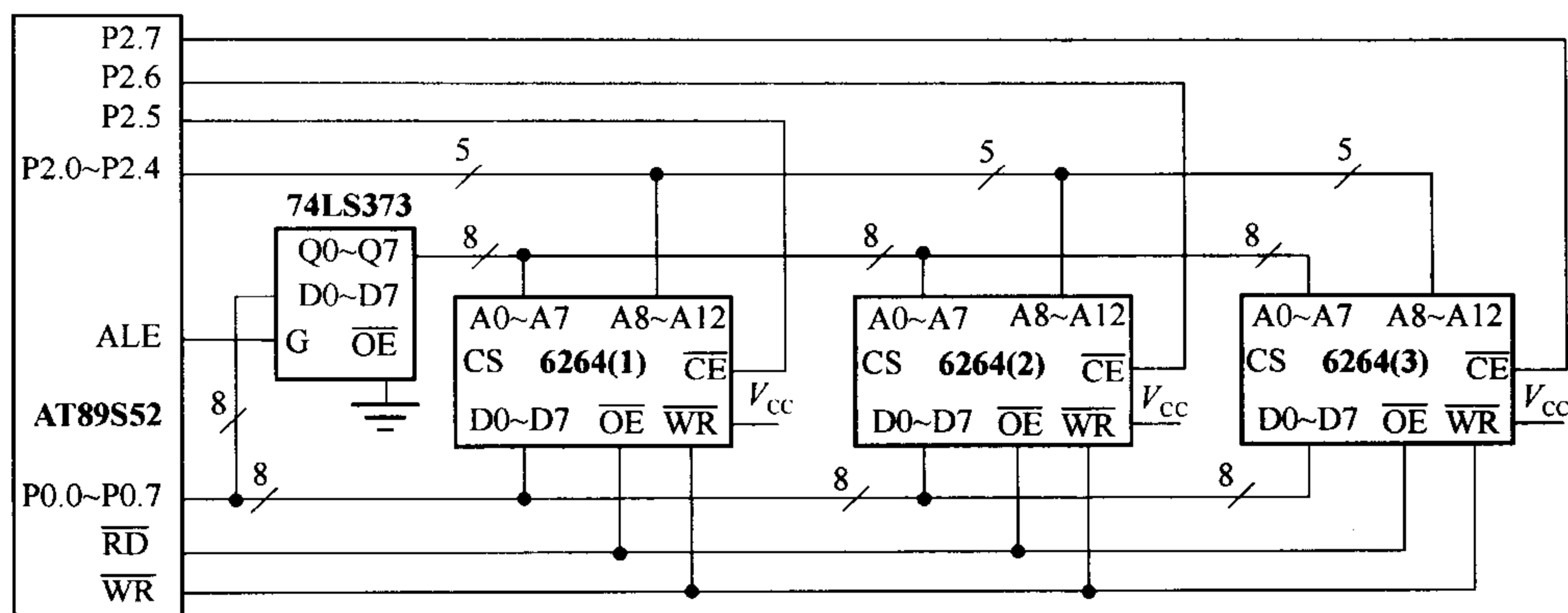


图 10.3.2 线选法扩展 3 片 6264 电路图

图 10.3.2 中数据存储器选用 6264,该芯片地址线为 A0~A12,故剩余地址线为 3 条。用线选法可扩展 3 片 6264,外部数据存储器空间可达 24 KB。第一片 6264 的地址范围为 0C000H~0DFFFH;第二片 6264 的地址范围为 0A000H~0BFFFH;第三片 6264 的地址范围为 6000H~7FFFH。

图 10.3.3 为用地址译码法扩展 AT89S52 外部数据存储器的电路图。图 10.3.3 中数据存储器选用 62128,该芯片地址线为 A0~A13,故剩余地址线为 2 根。若采用 2-4 译码器可扩展 4 片 62128,外部数据存储器总容量可达 64 KB。第一片 62128 的地址范围为 0000H~3FFFH;第二片 62128 的地址范围为 4000H~7FFFH;第三片 62128 的地址范围为 8000H~0BFFFH;第四片 62128 的地址范围为 0C000H~0FFFFH。

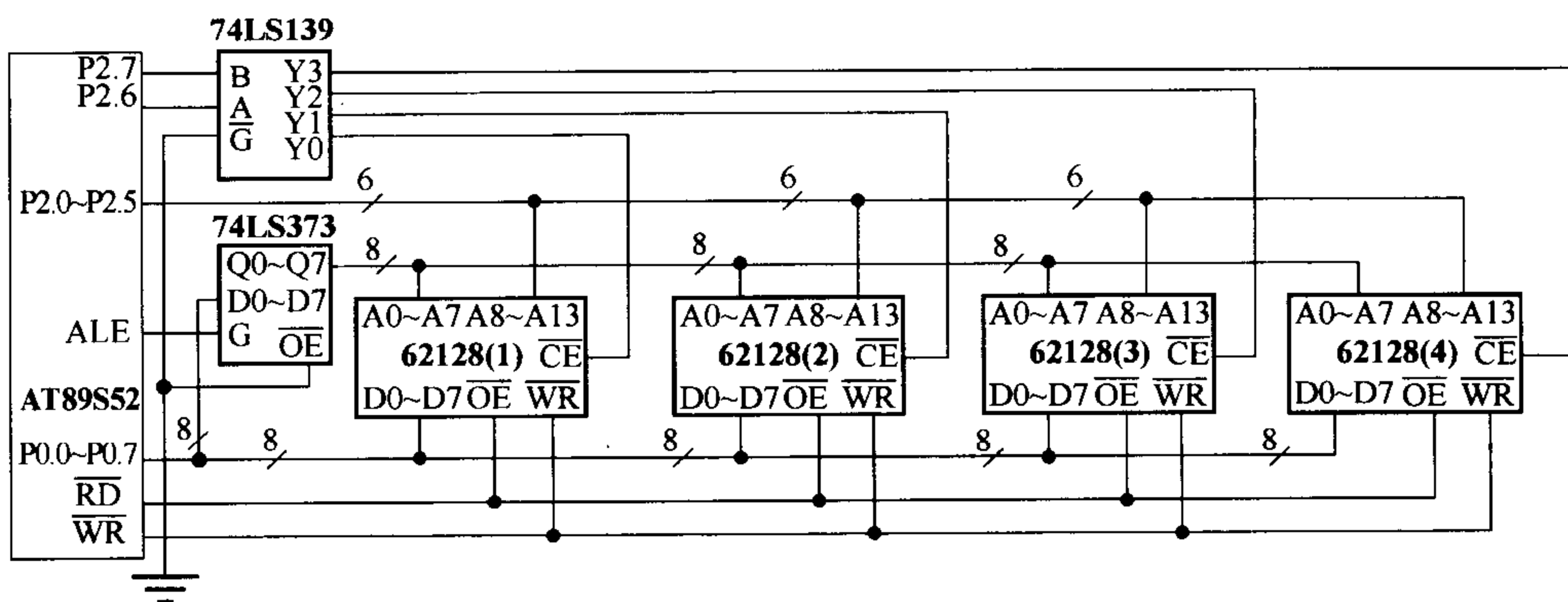


图 10.3.3 地址译码法扩展外部数据存储器电路图

根据上述电路,对外部 RAM 进行读写操作时的具体程序如下。

(1) 向 RAM 的 5000H 单元写数据 DATA:

```
MOV    A, #DATA
MOV    DPTR, #5000H
MOVX   @DPTR, A
```

(2) 从 7FFFH 单元读数据:

```
MOV    DPTR, #7FFFH
MOVX   A, @DPTR
```

### 10.3.2 并行 EEPROM 的扩展

EEPROM 是电擦除可编程只读存储器,其突出优点是能够在线擦除和改写,无须像 EPROM 那样必须用紫外线照射才能擦除。EEPROM 芯片在写入时能自动完成擦除,且不再需要专用的编程电源,可以直接使用单片机系统的 5 V 电源。在芯片的引脚设计上,2 KB 的 EEPROM 2816 与相同容量的 EPROM 2716 和静态 RAM 6116 兼容,8 KB 的 EEPROM 2864A 与同容量的 EPROM 2764A 和静态 RAM 6264 兼容。所以,这里把 EEPROM 归并到数据存储器类,当然也可以将其归并到 ROM 类。上述这些特点给硬件电路的设计和调试带来了不少方便。

程序调试时,用 EEPROM 代替 EPROM,既可以方便地修改程序,也能保存调试好的程序。与 RAM 芯片相比,EEPROM 的写操作速度较慢。另外,它的擦写有寿命限制,虽然有 1 万次之多,但也不宜用在数据频繁更新的场合。常用的并行 EEPROM 芯片有 2816/2816A、2817/2817A、2864A 等,其引脚如图 10.3.4 所示。

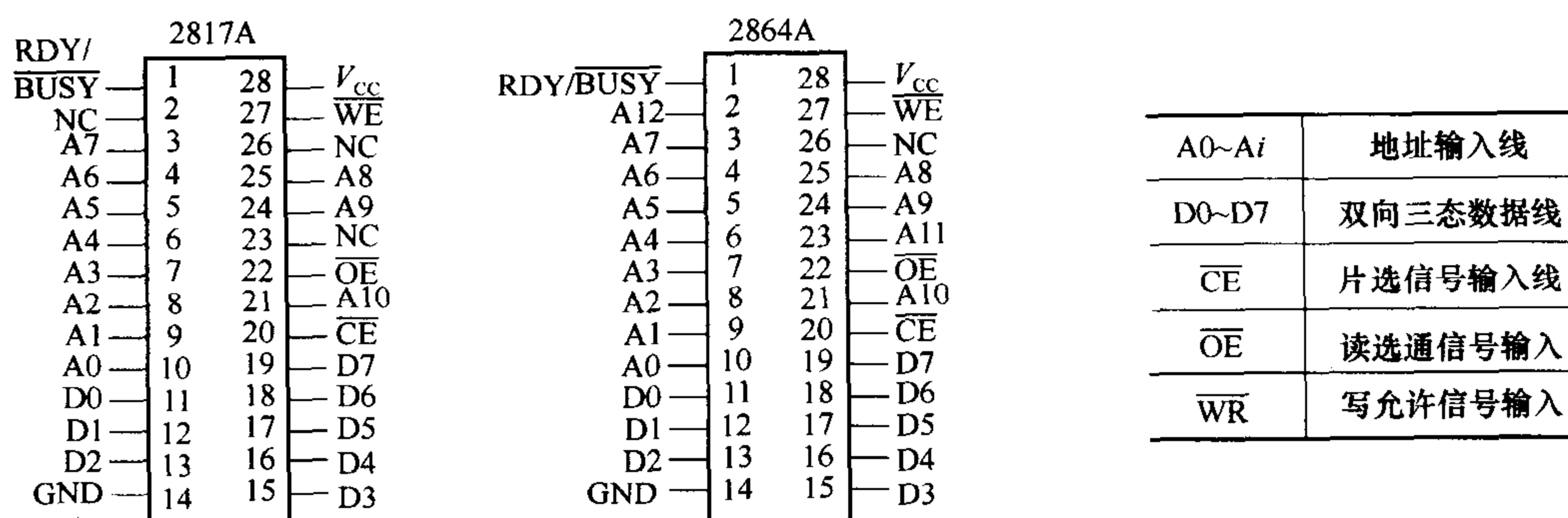


图 10.3.4 常用 EEPROM 引脚图

2817A 与 AT89S52 的连接电路如图 10.3.5 所示。

图 10.3.5 中,2817A 既可作外部数据存储器,又可作外部程序存储器使用。AT89S52 通过 P1.0 口线查询 2817A 的 RDY/ $\overline{BUSY}$ 脚的状态,来完成对 2817A 的写操作。2817A 的片选信号由 P2.7 提供。因 2817A 可作外部程序存储器和外部数据存储器合并使用,故将  $\overline{RD}$  和  $\overline{PSEN}$  信号加到一个与门上,并将其输出与 2817A 的数据输出允许

信号 $\overline{\text{OE}}$ 相连。

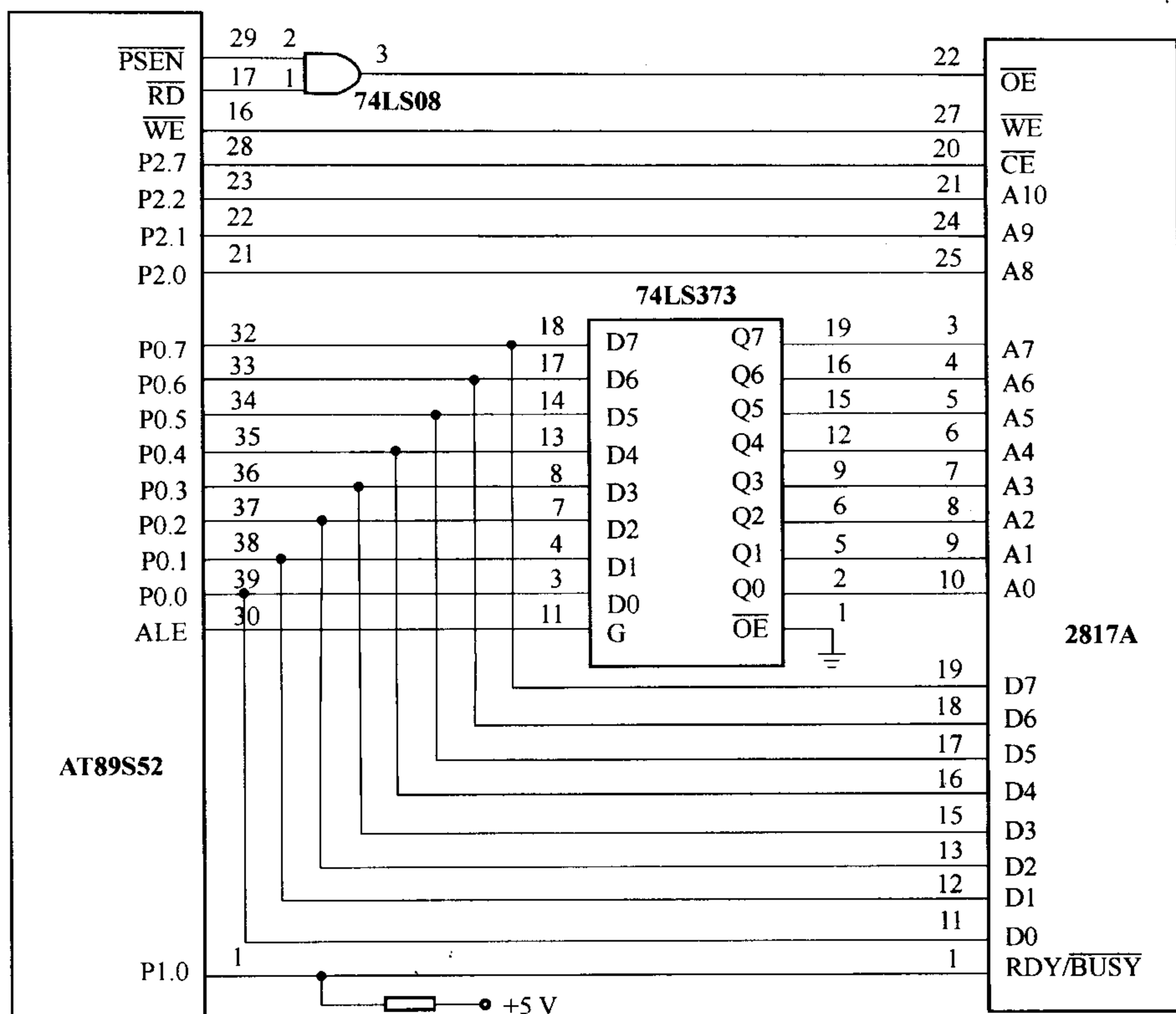


图 10.3.5 2817A 与 AT89S52 的连接电路图

**例 10-1** 根据图10.3.5中 2817A 与 AT89S52 的连接电路,编写对 2817A 进行写操作的子程序。要写入的数据取自源数据区。

子程序的入口参数如下。

R0: 写入的字节数计数器。

R1:2817A 的低 8 位地址寄存器。

R2:2817A 的高 8 位地址寄存器。

R3:源数据区首地址的低8位寄存器。

R4:源数据区首地址的高8位寄存器。

**解** 程序清单如下。

WR1:	MOV	DP0L,R3	
	MOV	DP0H,R4	;将源数据区 16 位地址传输到 DPTR0 中
	MOVX	A,@DPTR	;取数据
	INC	DPTR	;源数据地址指针加 1
	MOV	R3,DP0L	
	MOV	R4,DP0H	;将新的源数据区地址保存在 R3,R4 中
	MOV	DP0L,R1	
	MOV	DP0H,R2	;将 2817A 地址传输到 DPTR0 中



```

MOVX    @DPTR,A          ;将 A 的内容写入 DPTR0 中
WAIT:   JNB    P1.0, WAIT ;一个字节未写完等待
        INC    DPTR       ;2817A 地址加 1
        MOV    R1, DP0L
        MOV    R2, DP0H   ;将 2817A 的地址保存在 R1,R2 中
        DJNZ   R0, WR1    ;未完成,循环
        RET

```

### 10.3.3 串行 EEPROM 的扩展

X84041 是美国 Xicor 公司生产的 4 KB ( $512 \times 8$  bit) 容量 CMOS 工艺的串行 EEPROM。优点是低成本、低功耗、低电压运行、小封装尺寸,同时它具有较高的数据传输速率以及需要较少的接口代码。

#### 1. 特点

- 直接与单片机 CPU 接口,不需要 I/O 接口、接口逻辑和并行/串行转换器;
- 3.3 Mbit/s 数据传输速率;
- 低功耗 CMOS, 2.7~5.5 V 电压、待机电流小于  $50 \mu\text{A}$ 、工作电流小于 1 mA;
- 45 ns 读取时间。

写保护( $\overline{\text{WP}}$ )引脚提供了防止偶然对存储器进行写操作的硬件保护。

- 8 B 页写方式;
- 典型写周期时间: 5 ms;
- 高可靠性, 10 万次擦除次数、数据保存期 100 年;
- 8 引脚 DIP 和 SOIC、14 脚 TSSOP 封装。

#### 2. 结构和引脚

X84041 的内部结构和引脚如图 10.3.6 所示。

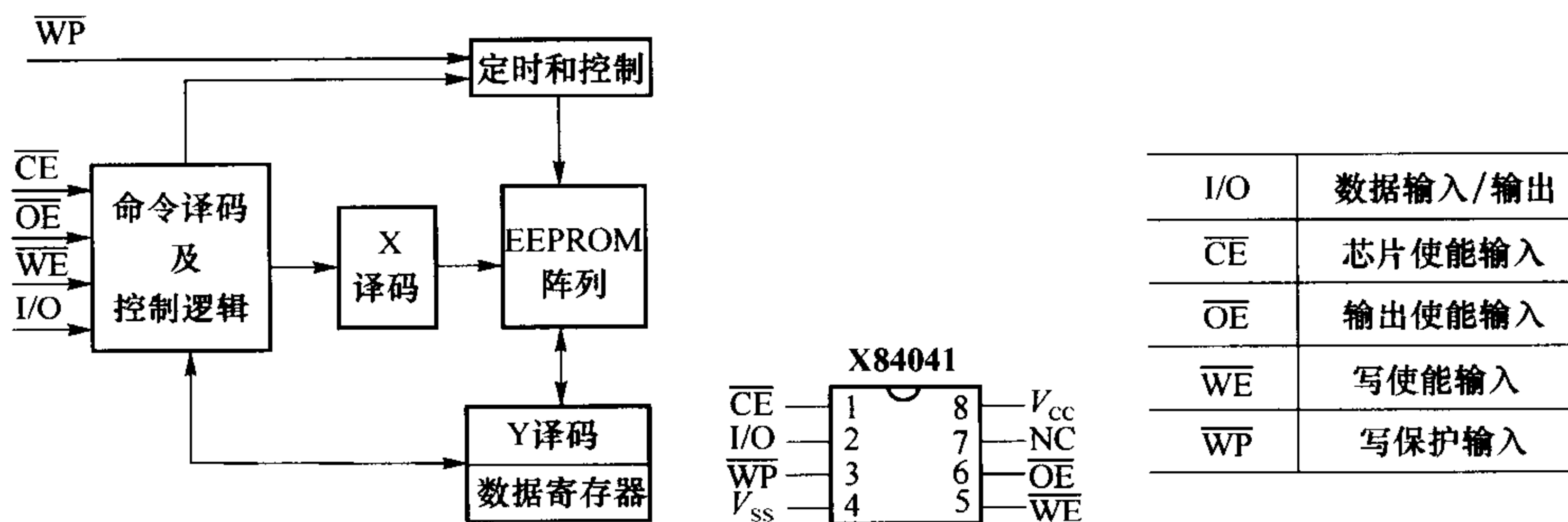


图 10.3.6 X84041 的内部结构和引脚

#### 3. X84041 的操作时序

用  $\overline{\text{CE}}$ 、 $\overline{\text{OE}}$  和  $\overline{\text{WE}}$  信号控制 X84041 的读和写操作, I/O 引脚用于串行发送和接收数据与命令。对 X84041 的操作按照操作时序进行, 包括数据定时、读操作、顺序读操作、复位

时序、写操作等过程。

#### (1) 数据定时

在 $\overline{\text{WE}}$ 或 $\overline{\text{CE}}$ 的上升沿(无论哪个先出现)锁存 I/O 引脚上输入的数据。当 $\overline{\text{WE}}$ 和 $\overline{\text{OE}}$ 均为低电平时, I/O 引脚上的输出数据有效。注意,  $\overline{\text{CE}}$ 为低电平时, 不能使 $\overline{\text{WE}}$ 和 $\overline{\text{OE}}$ 同时都变为低电平。

#### (2) 复位时序

复位时序将复位 X84041 片内各功能电路, 并设置内部写使能锁存器。在任何时候通过顺序执行“读周期、写 0 周期、读周期”的时序, 便可以向 X84041 发送一个复位时序。复位时序将中止连续读或写操作过程, 它通常作为读操作或写操作的开始。另外, 该复位时序可用来中断或结束顺序读或页装载(Page load)操作。此时序中的第二个读周期, 以及以后的任何读周期, 在 I/O 引脚上读到的都将是高电平, 直至出现有效的读周期为止。在读和写操作过程开始时, 必须发送复位时序以确保 X84041 对这些操作进行相应的初始化。

#### (3) 读操作时序

读时序如图 10.3.7 所示。每次读操作过程都要先发送一个 16 位的地址(实际只有其中的 9 位有用, 即 A8~A0), 其后才开始串行读取数据。地址的写入通过向器件发送 16 个独立的写周期( $\overline{\text{WE}}$ 和 $\overline{\text{CE}}$ 为低电平脉冲,  $\overline{\text{OE}}$ 为高电平)完成, 在写周期之间不能插入读周期。地址经 I/O 引脚串行送入, 最高有效位在前。一旦完成地址写入, 微处理器便可通过发送 8 个独立的读周期( $\overline{\text{OE}}$ 和 $\overline{\text{CE}}$ 为低电平脉冲,  $\overline{\text{WE}}$ 为高电平), 经 I/O 引脚从 X84041 中读取一个数据字节。此时, 若发送一个复位时序将会中止读操作, 否则, X84041 将处于顺序读出方式, 而等待下一步的读操作。在 X84041 的 $\overline{\text{CE}}$ 引脚为高电平期间, 微处理器可以单独执行总线上的其他任务。

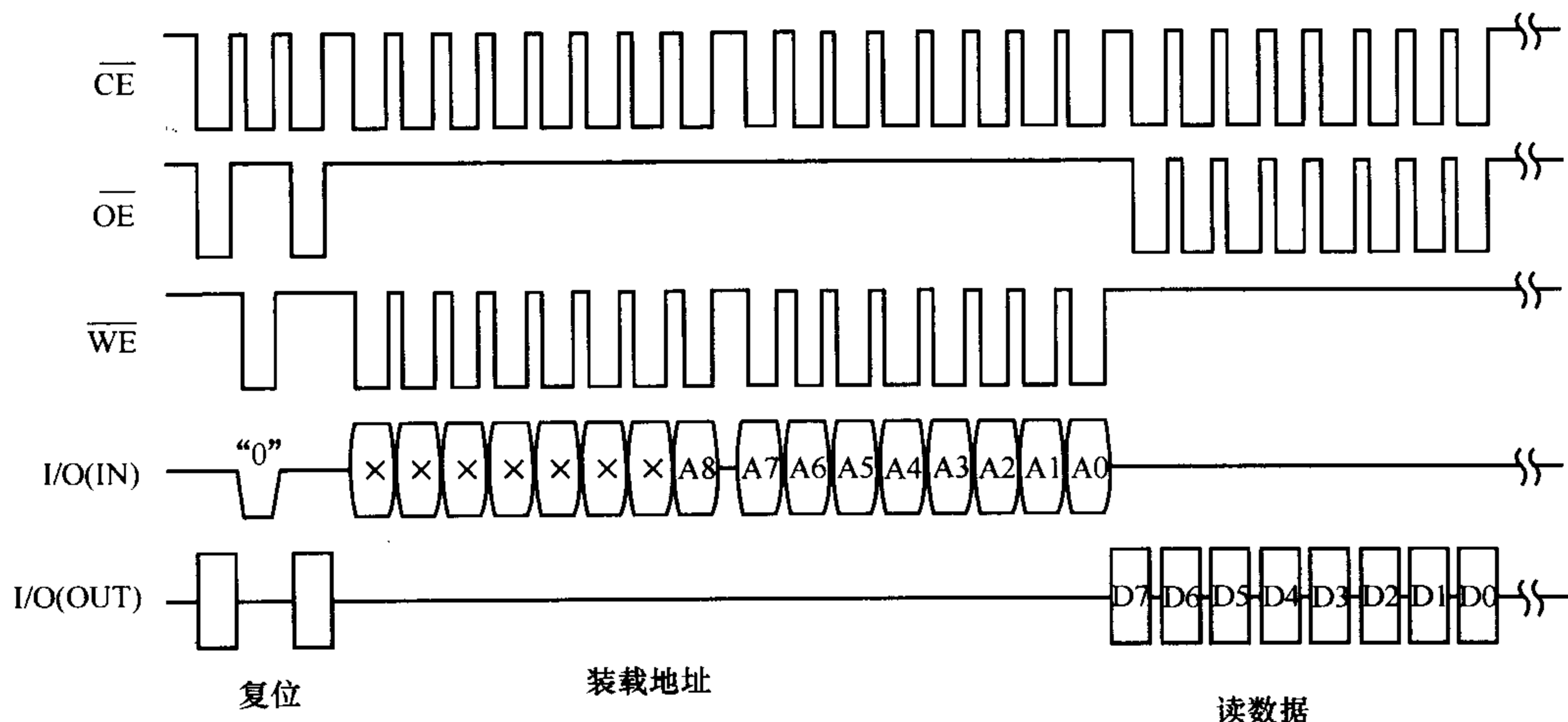


图 10.3.7 读操作时序

#### (4) 顺序读出方式

在读取每一个数据字节之后, 字节地址计数器自动加 1。通过继续发送读周期, 可以顺序读出存储器下一地址单元的数据字节。当地址达到最高地址 1FFH 时, 地址计数器

将会翻转至地址 000H, 读操作也将会无限地继续下去。

#### (5) 写操作时序

如图 10.3.8 所示, EEPROM 写操作过程(即对非易失性存储器进行写入操作的过程)包括:送入复位时序,送入 16 位地址(前 7 位未用),送入 1~8 个数据字节(称为页装载);一个特定的开始 EEPROM 写命令时序;将数据字节逐个写入 EEPROM。

发送复位时序的目的是用来设置内部写使能锁存器。通过向 X84041 发送 16 个独立的写周期,串行写入地址  $\times\times\times\times\times\times\times\times A8A7A6A5A4A3A2A1A0H$ 。通过发送 8, 16, 24, 32, 40, 48, 56 或 64 个独立的写周期,可写入 1 B, 2 B, 3 B, 4 B, 5 B, 6 B, 7 B 或 8 B 的数据。并且,在写周期之间不允许插入读周期。

特定的开始 EEPROM 写命令时序目的是对 EEPROM 写周期进行初始化。通过顺序发送“读周期,写 1 周期,读周期”这一特定的时序完成。第 1 个读周期结束页装载操作,其后的写 1 周期和读周期,将开始 EEPROM 写操作。X84041 可以识别从地址  $\times\times\times\times\times\times\times\times 000H$  开始的一页内的 8 B。当向器件装载过多的数据,并且地址超出页的上限地址时,将会使地址计数器回复到该页的起始地址,而在此地址上继续装入数据。由于这一原因,发送大于 64 个连续的数据位(即 8 B)将导致重写并覆盖先前的数据。如果发出的写周期不完整,那么不能开始 EEPROM 写周期。当 EEPROM 写周期完成之后,内部的写使能锁存器自动复位,以防止偶然性的意外写操作。同样,当  $\overline{CE}$  为高电平时,处理器可单独地执行总线上的其他任务。

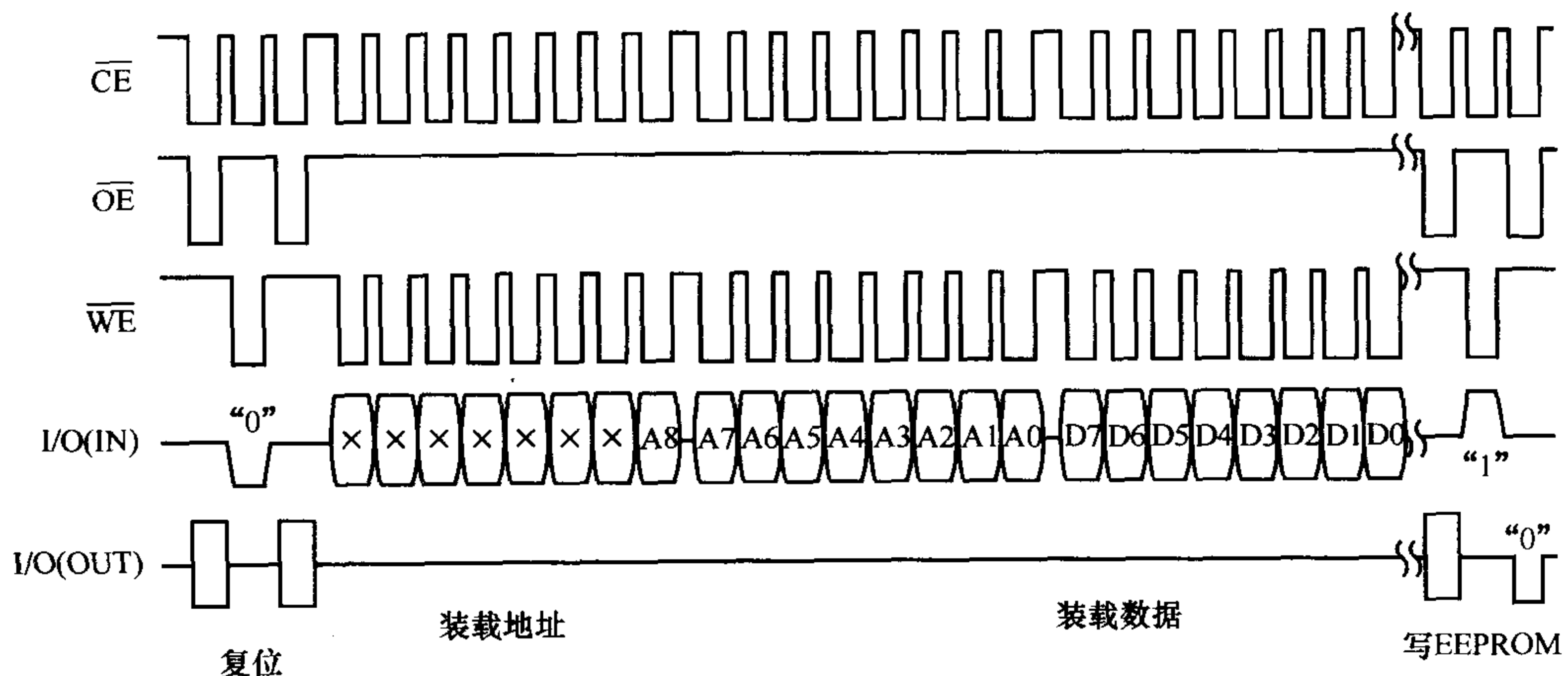


图 10.3.8 写操作时序

#### (6) EEPROM 写状态

任何时候通过简单地读取 X84041 的 I/O 引脚的状态,都可以判断 EEPROM 写周期的状态。在  $\overline{OE}$  和  $\overline{CE}$  为低电平而  $\overline{WE}$  为高电平时读取此引脚,当 EEPROM 写周期尚未完成, I/O 引脚为低电平;而当 EEPROM 写周期完成时, I/O 引脚变为高电平。在 EEPROM 写周期内也可以发送复位时序,但得到的结果是同样的,即,只要 EEPROM 写周期正在进行, I/O 引脚便是低电平,而 EEPROM 写周期完成后, I/O 引脚为高电平。

#### (7) EEPROM 写保护

X84041 器件在电路上设置了下列功能以防止意外写操作给数据安全性带来的影响:

- 上电时复位内部写使能锁存器;
- 在开始写时序之前为了设置内部写使能锁存器必须发送复位时序;
- 为了开始 EEPROM 写周期,需要特定的开始 EEPROM 写命令时序;
- 在 EEPROM 写周期的末尾自动复位内部写使能锁存器;
- 只要  $\overline{\text{WP}}$  引脚为低电平,内部写使能锁存器就被复位且保持复位状态,这将禁止所有的 EEPROM 写操作。

#### 4. AT89S52 单片机与 X84041 接口及程序

X84041 和 AT89S52 单片机的连接如图 10.3.9 所示, X84041 的片选信号  $\overline{\text{CE}}$  由 74LS138 产生。AT89S52 单片机通过  $\overline{\text{RD}}$  和  $\overline{\text{WR}}$  控制 X84041 的读写过程, 利用 MOVX 指令产生 X84041 所需的读、写控制信号  $\overline{\text{OE}}$  和  $\overline{\text{WE}}$ 。X84041 的片内地址由 I/O 数据线在内部时钟同步下串行输入, 通过内部译码选中相应地址单元, 再串行传送数据。

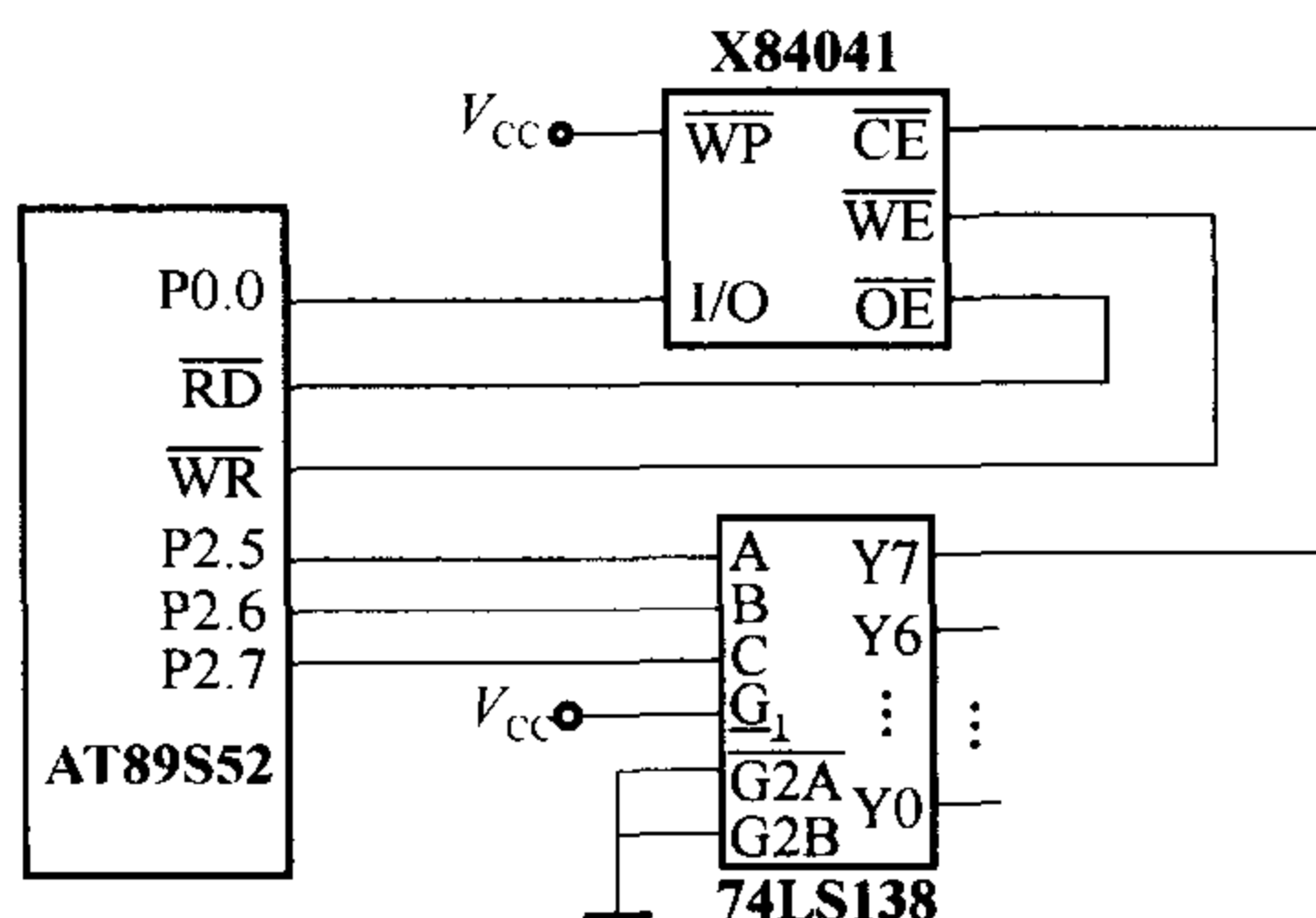


图 10.3.9 X84041 和 AT89S52 的连接

X84041 读写数据的流程如图 10.3.10 所示。如果是连续多字节的读写, 则只需先发首地址, X84041 自动修改地址。

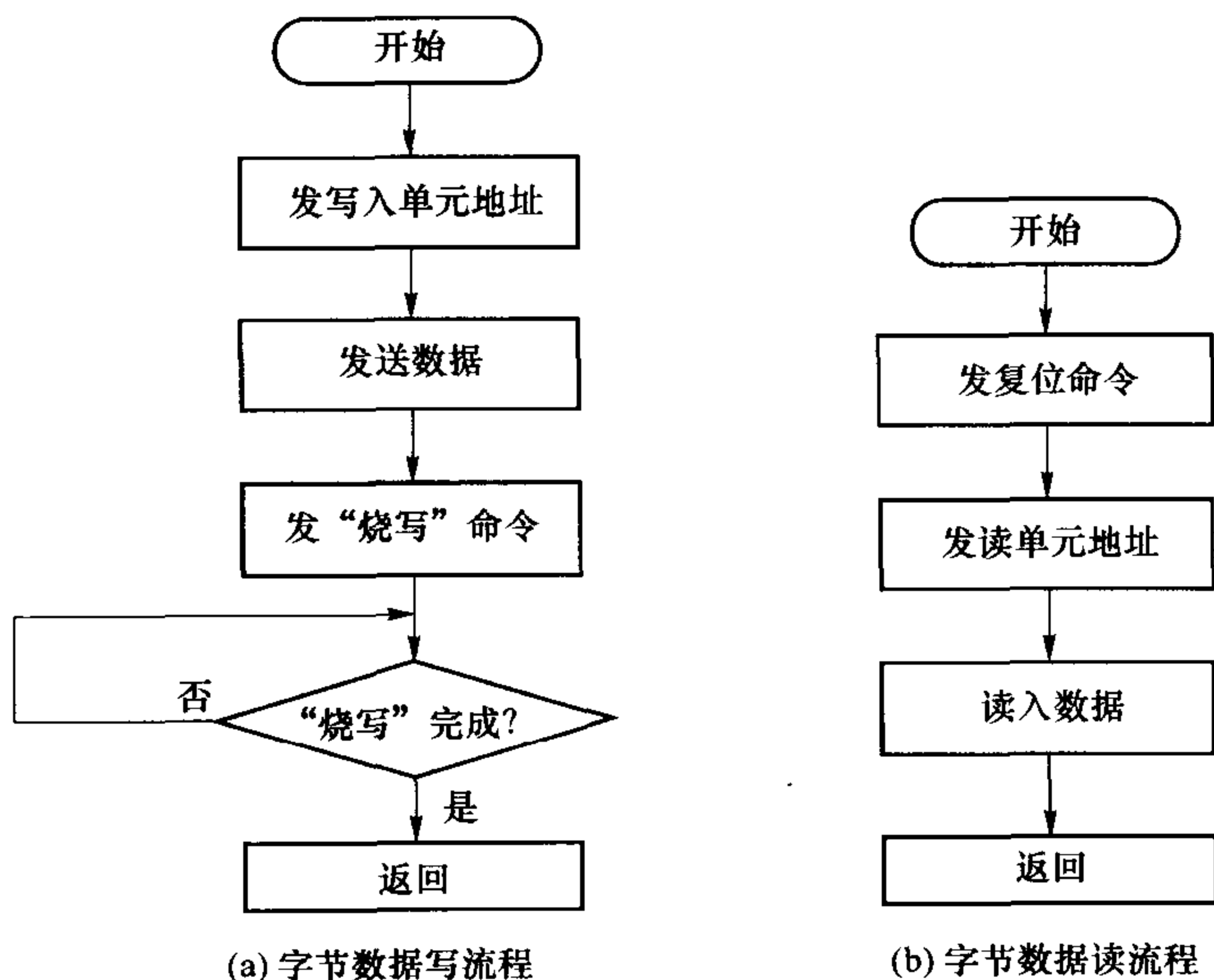


图 10.3.10 X84041 的读写流程

读写 X84041 程序清单如下。

字节写子程序：

```

BYTE_WRITE:    MOV     DPTR, #BYTE_ADDR    ;送字节地址到 DPTR
                LCALL   RST                ;复位 X84041
                MOV     A, DP0H
                LCALL   LOAD_BYTE           ;发送高位地址
                MOV     A, DP0L
                LCALL   LOAD_BYTE           ;发送低位地址
                MOV     A, #BYTE_DATA      ;发送数据到 A
                LCALL   LOAD_BYTE           ;发送数据字节
                LCALL   NV_START            ;发烧写命令
                LCAKK   NV_POLL            ;调查询子程序

```

字节读子程序：

```

BYTE_READ:     MOV     DPTR, #BYTE_ADDR    ;送字节地址到 DPTR
                LCALL   RST                ;复位 X84041
                MOV     A, DP0H
                LCALL   LOAD_BYTE           ;发送高位地址
                MOV     A, DP0L
                LCALL   LOAD_BYTE           ;发送低位地址
                LCALL   REC_BYTE            ;调字节读入子程序
                RET

```

被调用子程序如下。

复位子程序：

```

RST:           MOVX    A, @DPTR            ;读周期,  $\overline{OE} = 0, \overline{WE} = 1$ 
                MOV     A, #00H            ;输出起始位 0
                MOVX    @DPTR, A           ;写“0”周期,  $\overline{OE} = 1, \overline{WE} = 0, P0.0 = 0$ 
                MOVX    A, @DPTR            ;读周期,  $\overline{OE} = 0, \overline{WE} = 1$ 
                RET

```

发送数据子程序：

```

LOAD_BYTE:     MOV     R0, #08            ;发送 8 位数据
LOOP1:         RL      A                  ;数据左移选送最高位
                MOVX    @DPTR, A
                DJNZ    R0, LOOP1          ;8 位数据未发完继续
                RET

```

发烧写命令子程序：

```

NV_START:      MOVX    A, @DPTR            ;读周期
                MOV     A, #01H
                MOVX    @DPTR, A           ;发 1 使  $\overline{WE}$  失效, 写“1”周期,  $P0.0 = 1$ 

```

```

MOVX    A, @DPTR        ;读周期
RET

```

查询烧写是否完成子程序:

```

NV_POLL:  MOV    R0, #0FFH
LOOP3:    MOVX    A, @DPTR        ;检测写周期是否完成
          NOP
          NOP
          NOP
          JB     ACC.0, END_LOOP
          DJNZ   R0, LOOP3
END_LOOP:  RET
REC_BYTE:  MOV    R7, #08        ;字节读入子程序
          MOV    R1, #00
LOOP2:    CLR     C
          MOVX    A, @DPTR
          JNB    ACC.0, ZERO_BIT
          SETB   C
ZERO_BIT:  MOV    A, R1
          RLC     A
          MOV    R1, A
          DJNZ   R7, LOOP2
          MOV    A, R1
          RET

```

## 10.4 并行 I/O 接口的扩展

AT89S52 单片机本身提供给用户使用的输入、输出口为 P0~P3,但当外部扩展程序存储器或数据存储器后,只有 P1 和部分 P3 口线作通用 I/O 接口使用,因此,根据单片机应用系统需要,有时要进行 I/O 口的扩展。

I/O 接口和扩展 I/O 接口可以解决以下问题:

- (1) 高速 CPU 与工作速度快慢差别很大的慢速外围设备的矛盾;
- (2) 高速 CPU 与外围设备之间的数据格式转换和匹配问题;
- (3) CPU 的负载能力和外围设备端口选择问题。

### 10.4.1 简单 I/O 接口的扩展

74LS244 是总线驱动器,它带负载能力较强可作扩展输入。74LS373(8-D 锁存器)作扩展输出。它们可以直接与 P0 口相连。

图 10.4.1 中, P0 口为双向数据线, 既能从 74LS244 输入数据, 又能将数据传送给 74LS373 输出。输出控制信号由 P2.0 和  $\overline{WR}$  合成。当两者同时为 0 电平时, 或非门输出 1, 将 P0 口数据锁存到 74LS373, 其输出控制发光二极管 LED, 当某线输出 0 电平时, 该线上的 LED 发光。

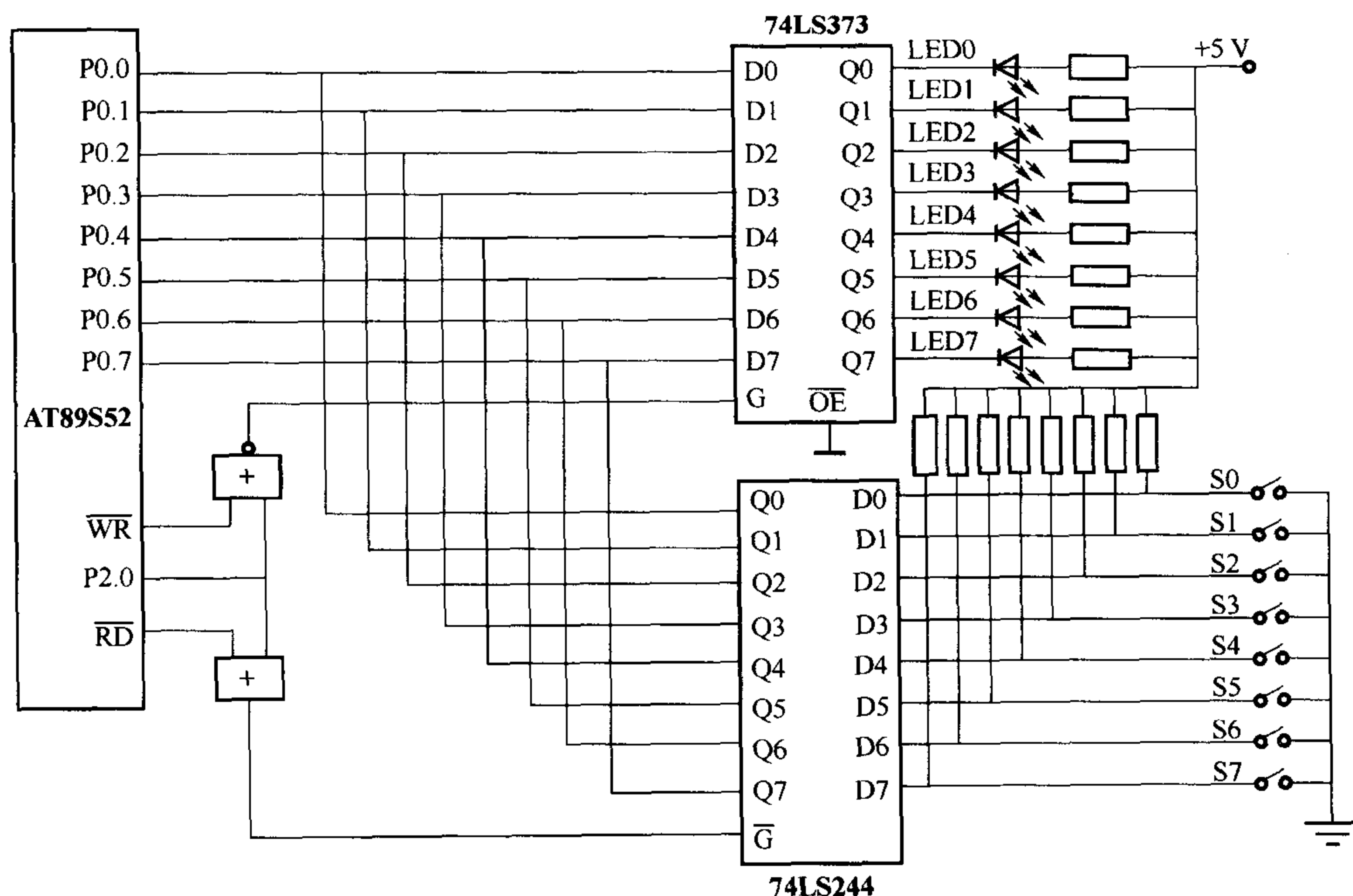


图 10.4.1 简单 I/O 接口扩展举例

输入控制信号由 P2.0 和  $\overline{RD}$  合成。当两者同时为 0 电平时, 或门输出 0, 选通 74LS244, 将外部信号输入到总线。无键按下时, 输入为全 1; 若按下某键, 则所在线输入为 0。

输入和输出都是在 P2.0 为 0 时有效, 74LS244 和 74LS373 的地址都为 FEFFH。

图 10.4.1 电路的功能是按下任意键, 对应的 LED 发光。程序如下:

```

LOOP:   MOV     DPTR, #0FEFFH      ; 扩展 I/O 接口地址送 DPTR
        MOVX    A, @DPTR           ; 通过 244 读入数据, 检测键的状态
        MOVX    @DPTR, A           ; 向 373 输出数据, 驱动 LED
        SJMP    LOOP              ; 循环
  
```

## 10.4.2 可编程 8155 的并行 I/O 扩展

Intel 8155 芯片内具有 256 B 静态 RAM, 两个 8 位、一个 6 位的 I/O 接口和一个 14 位“减 1”计数器, 最大存取时间为 400 ns。8155 可直接和单片机连接, 不需要增加任何硬件逻辑电路, 是单片机系统中最常用的外围接口芯片之一。



### 1. 8155 的组成结构及引脚

8155 的组成结构及 40 脚双列直插式封装引脚如图 10.4.2 所示。

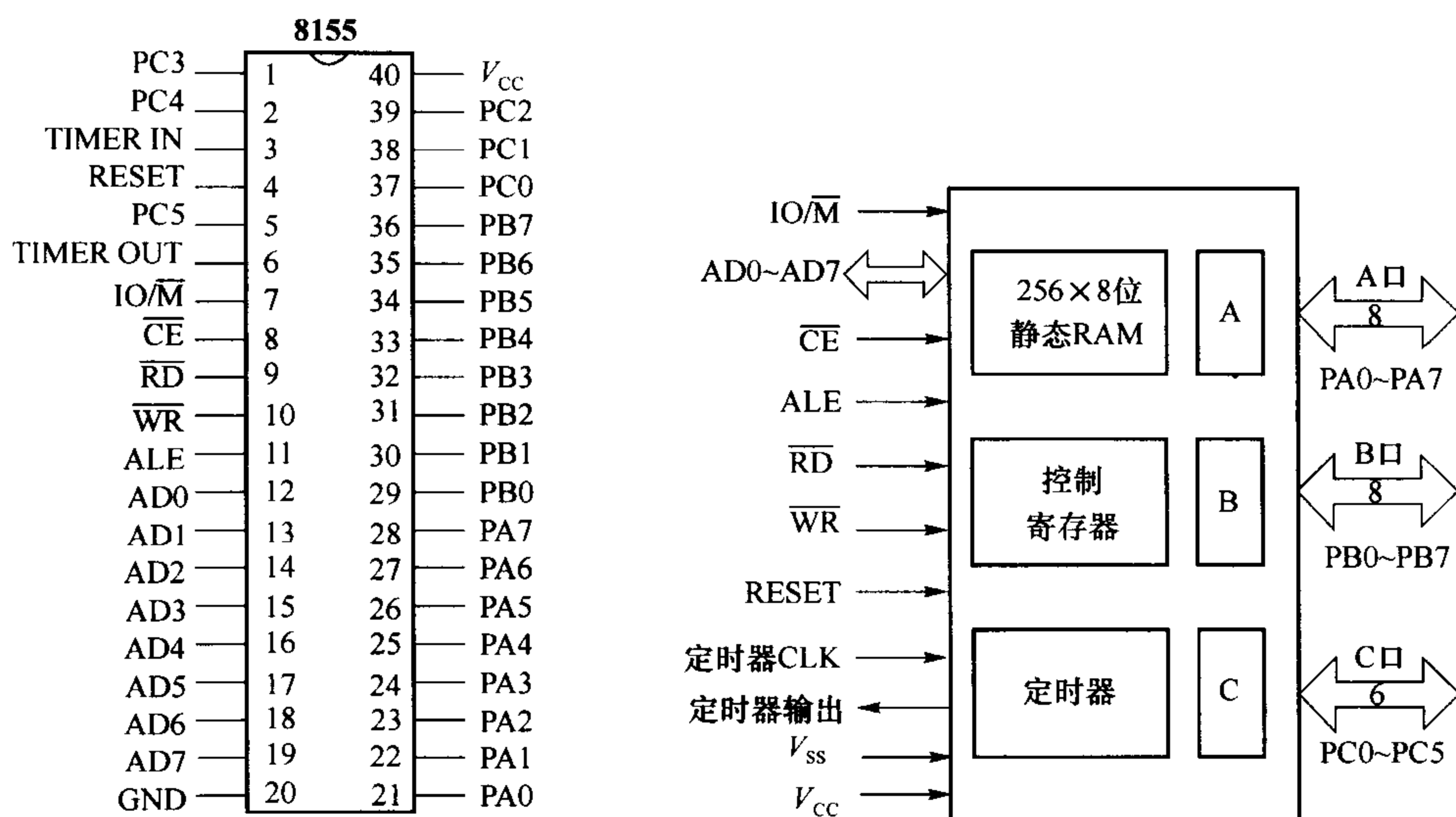


图 10.4.2 8155 的组成结构框图及其芯片的引脚

8155 的引脚及功能说明如下。

**RESET:** 复位信号线。该端出现  $5\mu s$  左右的正脉冲时, 8155 被初始复位, 复位后 I/O 接口设置为输入方式。该引脚一般可与 AT89S52 的 RST 端相连, 即 8155 与单片机同时复位。

**AD7~AD0:** 地址数据总线, 它的分时复用功能和 AT89S52 的 P0 口完全相同。用来传送单片机和 8155 之间的地址、数据、命令和状态信息。根据  $\overline{IO/\overline{M}}$  引脚的状态将决定地址总线上是 8155 的 RAM 单元地址或 I/O 接口地址。 $\overline{RD}$  有效还是  $\overline{WR}$  有效将决定是从 8155 读出数据还是要把数据写入 8155。

**$\overline{CE}$ :** 片选信号, 低电平有效。

**$\overline{IO/\overline{M}}$ :** IO 接口与存储器选择信号, 高电平选择 I/O 接口, 低电平选择存储器。

**$\overline{RD}$ :** 读选通信号, 低电平有效。配合  $\overline{IO/\overline{M}}$  决定是读 I/O 接口数据还是读片内存储器数据到 AD 总线。

**$\overline{WR}$ :** 写选通信号, 低电平有效。配合  $\overline{IO/\overline{M}}$  决定是把 AD 总线数据写到 I/O 接口还是写到片内存储器。

**ALE:** 地址及片选锁存信号, 高电平有效, 下降沿将地址及片选信号锁存到芯片中。

**PA0~PA7:** 端口 A 的 I/O 口线。由程序控制的命令寄存器选择输入输出方向。

**PB0~PB7:** 端口 B 的 I/O 口线。由程序控制的命令寄存器选择输入输出方向。

**PC0~PC5:** 端口 C 的口线。有两个作用, 一是作为 C 口的 I/O 口线; 二是作为 A 口和 B 口的控制联络信号线, 由命令寄存器进行控制。作控制信号线时, 其功能如下。

PC0, A INTR(A 口中断申请信号线)。

PC1, A BF(A 口缓冲器满信号线)。

PC2, A STB(A 选通线)。

PC3, B INTR(B 口中断申请信号线)。

PC4, B BF(B 口缓冲器满信号线)。

PC5, B STB(B 选通线)。

TIMER IN: 定时/计数器输入端。

TIMER OUT: 定时/计数器输出端。

## 2. 8155 的 RAM 和 I/O 接口地址编码

AT89S52 单片机应用系统中 8155 的 RAM 和 I/O 接口按外部数据存储器统一编址。

当  $IO/\overline{M}=0$  时, 单片机对 8155 RAM 进行读/写, RAM 低 8 位地址为 00H~FFH;

当  $IO/\overline{M}=1$  时, 单片机对 8155 中的 I/O 接口进行读/写。8155 内部 I/O 及定时器的低 8 位地址如表 10.4.1 所示。

表 10.4.1 8155 I/O 接口地址编码及功能表

地 址	引 出 端	功 能
×××××000	内部	命令寄存器(只写)
×××××000	内部	状态寄存器(只读)
×××××001	PA0~PA7	A 口
×××××010	PB0~PB7	B 口
×××××011	PC0~PC5	C 口或控制联络线
×××××100	内部	定时/计数器低 8 位寄存器
×××××101	内部	定时/计数器高 6 位寄存器和定时/计数器输出波形工作方式字

其中, 命令寄存器和状态寄存器共用一个地址, 所以又称为命令/状态寄存器, 由读/写指令来区分是写命令寄存器还是读状态寄存器。

如: MOVX     A, @DPTR                                 ; 读状态寄存器

      MOVX     @DPTR, A                                ; 写命令寄存器

## 3. 8155 的 I/O 接口工作方式及选择

### (1) 命令寄存器

8155 可编程接口芯片只有一个控制字。将一个 8 位控制字写入命令寄存器, 就可以确定 PA 口、PB 口、PC 口和定时器的工作方式及功能, 如图 10.4.3 所示。

从工作方式控制字可以看出, 其低 4 位(D3~D0)用以确定 I/O 接口的工作方式; 当 PC 口被编程为控制信号工作方式(ALT3、ALT4)时, D5 和 D4 位用来确定允许/禁止从 C 口输出中断申请信号, D7 和 D6 位是定时/计数器的工作方式控制位。

### (2) I/O 接口的工作方式

- 基本输入输出: 当 8155 被编程为 ALT1、ALT2 时, A 口、B 口、C 口均工作于基本输入输出方式。其区别是 ALT1 时, C 口为输入; ALT2 时, C 口为输出。
- 选通输入输出: 当 8155 被编程为 ALT3 时, A 口定义为选通 I/O, B 口定义为基本 I/O; 当 8155 被编程为 ALT4 时, A 口、B 口均定义为选通 I/O 方式。ALT4 的逻

辑组态如图 10.4.4 所示。

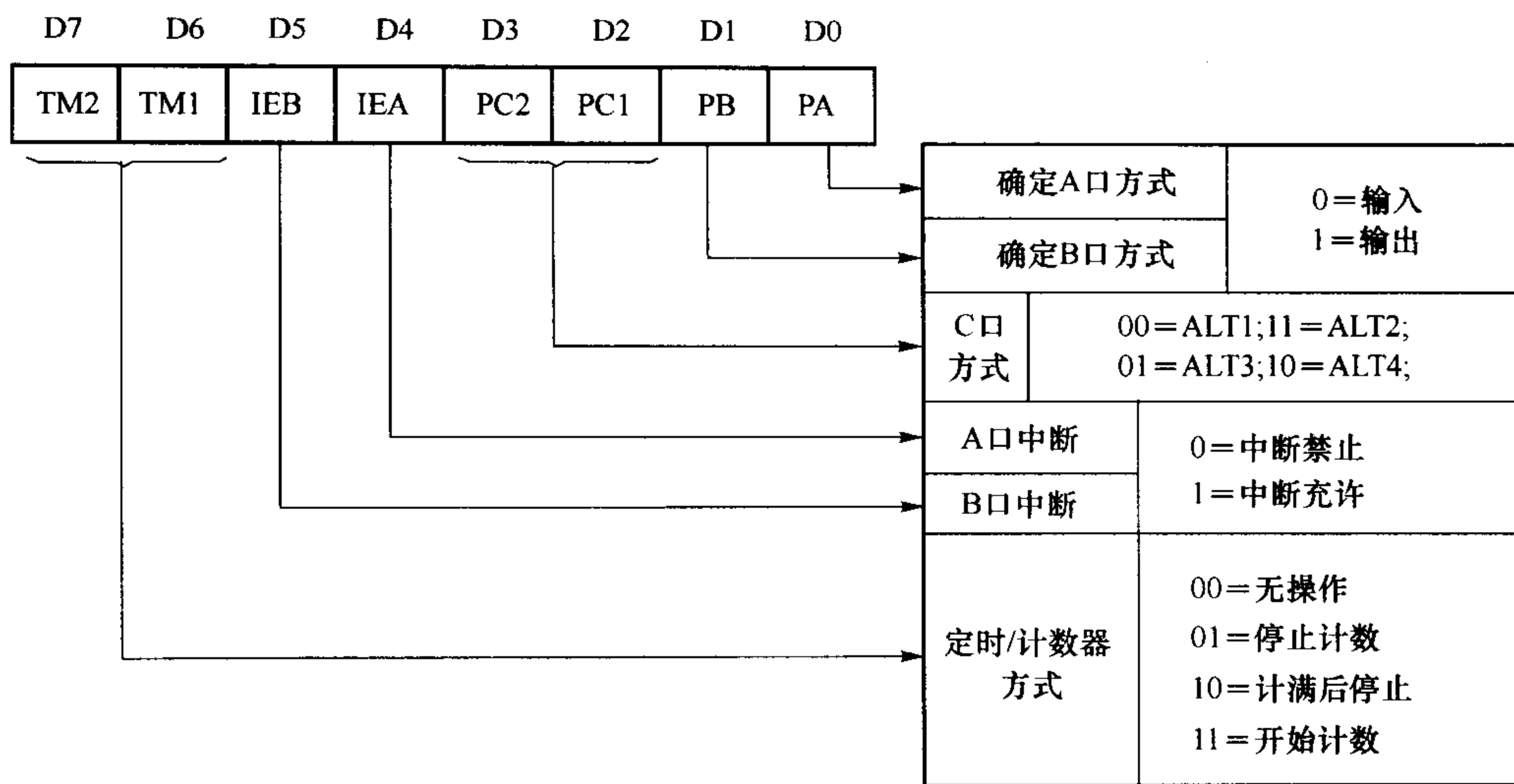


图 10.4.3 8155 命令寄存器的工作方式控制字格式

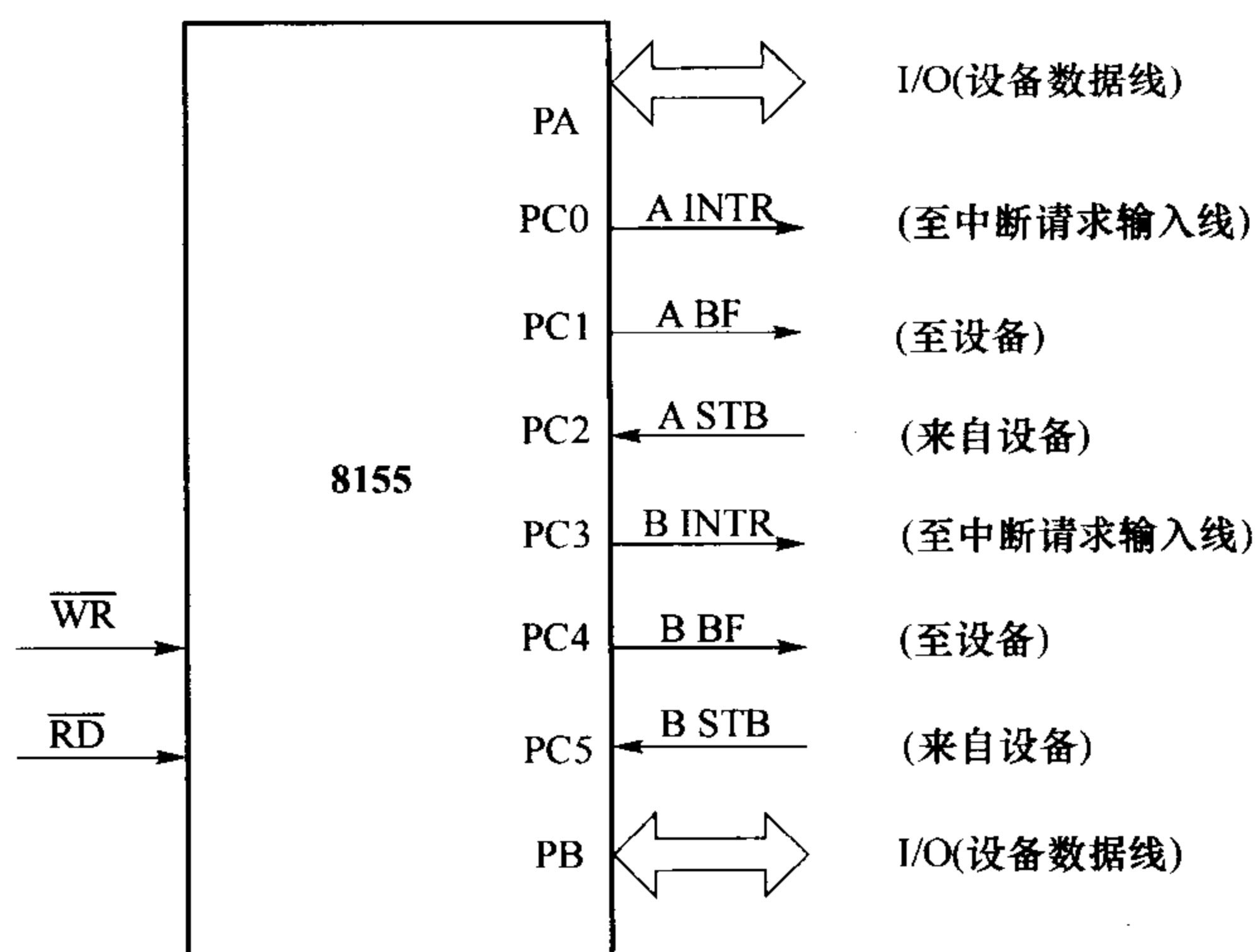


图 10.4.4 8155 的 A 口、B 口均为选通输入输出时的逻辑结构

INTR 为中断请求输出线,作为 CPU 的中断源,高电平有效。当 8155 的 A 口(或 B 口)缓冲器接收到设备输入的数据,中断请求线升高(条件是命令寄存器中相应中断允许位为 1 时),向 CPU 申请中断,CPU 对 8155 相应的 I/O 口进行一次读/写操作,之后 INTR 变为低电平。

BF 为 I/O 缓冲器状态标志输出线。缓冲器存有数据时,BF 为高电平;缓冲器空时,BF 为低电平。

STB 为设备选通信号输入线,低电平有效。

### (3) 状态寄存器及 I/O 接口的状态字

8155 有一个状态寄存器,状态字的格式如图 10.4.5 所示。状态寄存器由 7 位锁存器组成,每位是一个状态。该寄存器是只读寄存器。在对 8155 读操作时,能从 I/O 地址  $\times\times\times\times\times000\text{B}$  读出。它锁存 I/O 口和定时器的当前工作状态,供 CPU 查询。

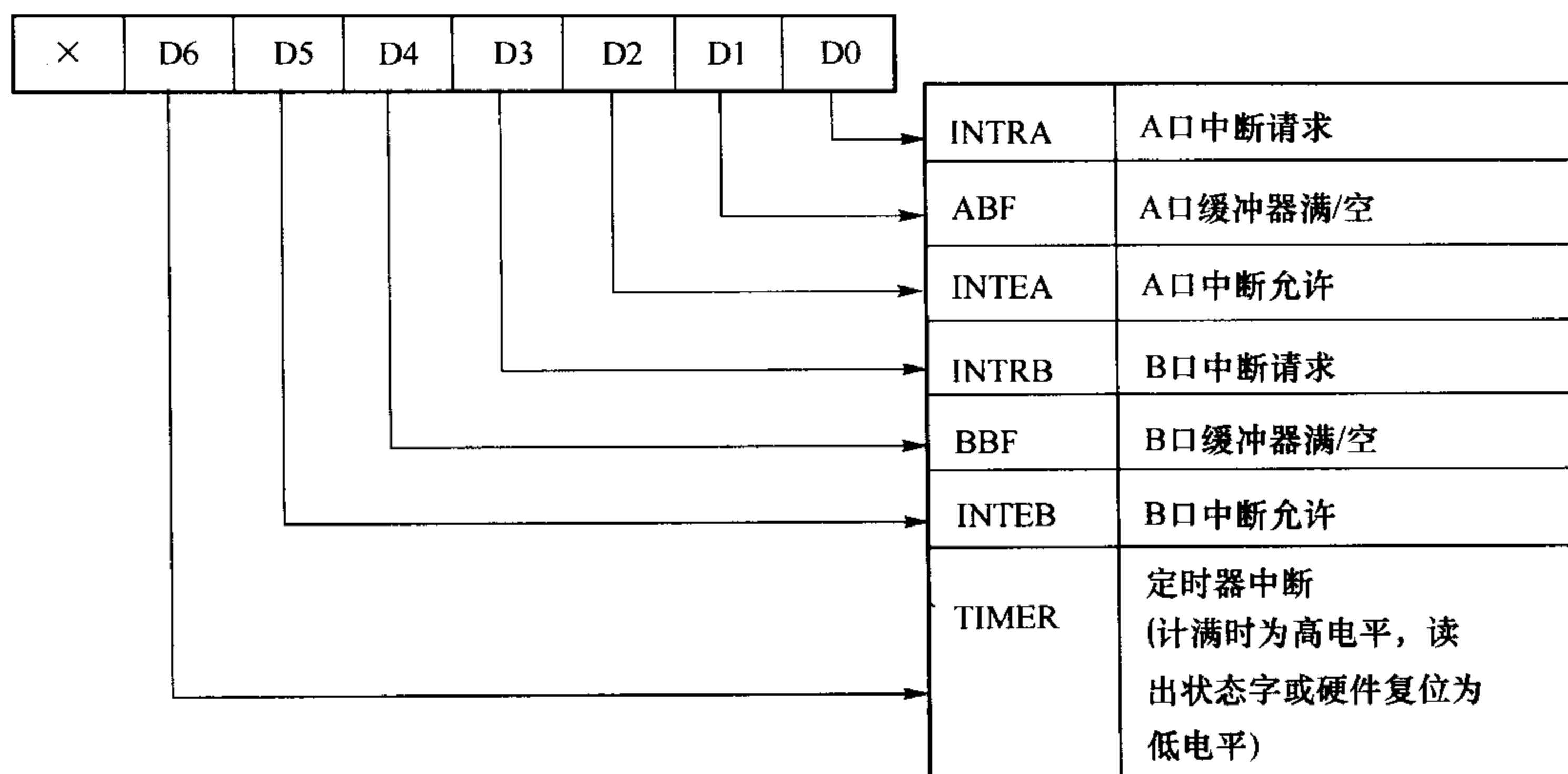


图 10.4.5 8155 状态寄存器格式

### (4) PA 寄存器及 A 口

可以按照命令寄存器的内容确定是输入还是输出寄存器,也可按照命令使这个口既可在基本方式下工作,又可以在联络线的选通方式下工作。

### (5) PB 寄存器及 B 口

与 PA 寄存器的功能相同。

### (6) PC 寄存器及 C 口

PC 寄存器仅有 6 位,由命令寄存器的第二位和第三位确定。这 6 位可以为基本输入口、输出口或作为 PA 和 PB 接口的控制信号。当 PC 作为控制口时,PC0~PC2 位分配给 PA 口,PC3~PC5 位分配给 PB 口。

## 4. 8155 的定时/计数器

### (1) 定时器方式选择

在 8155 中还设置有一个 14 位的定时/计数器,可用来定时或对外部事件计数,CPU 可通过程序选择计数长度和计数方式。计数长度和计数方式由输入到计数寄存器的计数控制字来确定,计数寄存器的格式如图 10.4.6 所示。

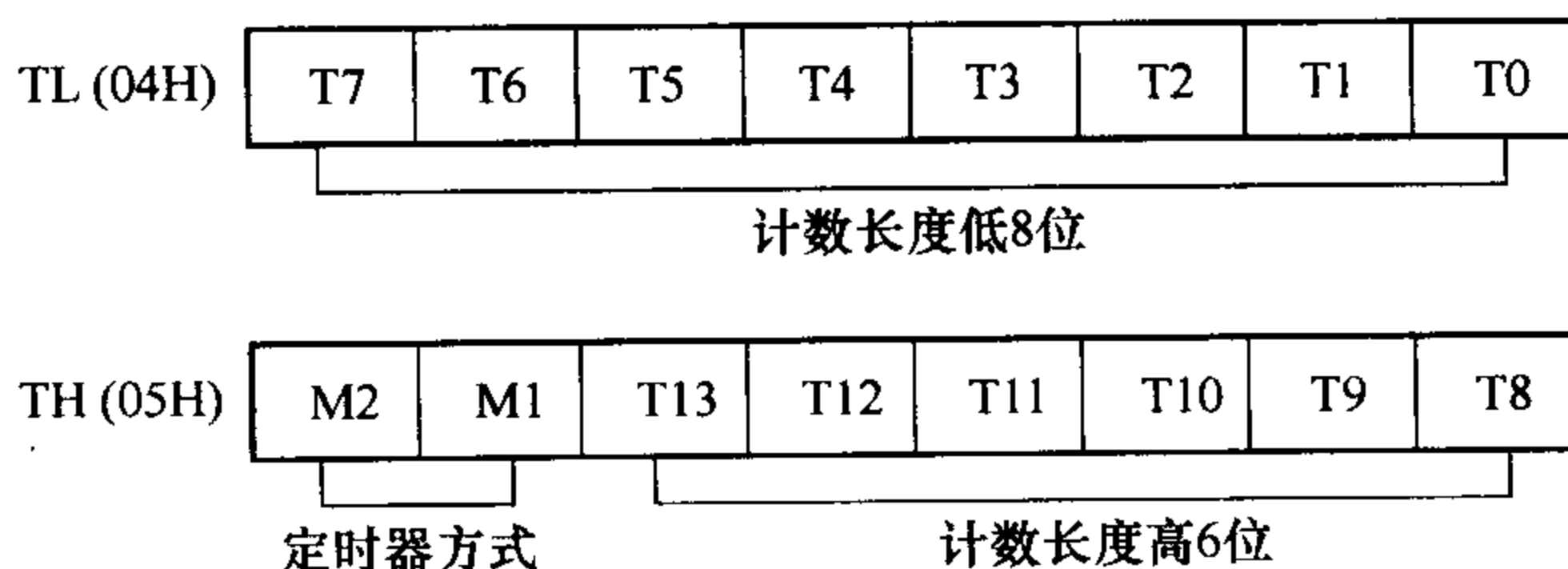


图 10.4.6 8155 定时器格式

其中 T0~T13 为计数长度,可表示的长度范围为 2H~3FFFH。TH 的最高两位 M2、M1 用来设置定时器的输出方式。8155 定时器方式及相应的输出波形如图 10.4.7 所示。

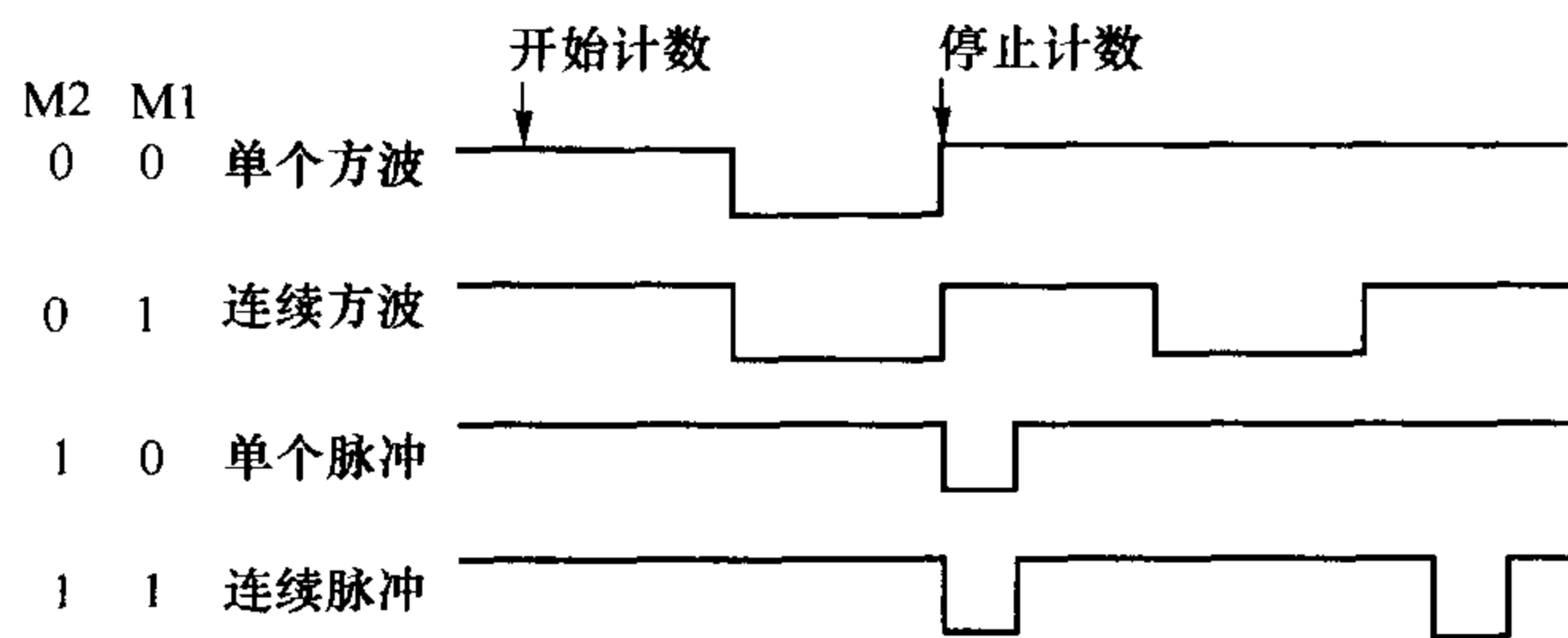


图 10.4.7 8155 定时器方式及输出波形

## (2) 定时器编程

对定时器进行编程时,首先将计数常数及定时器方式送入计数寄存器 TL 及 TH;然后写启动定时器命令到命令寄存器。

任何时候都可以设置定时器的长度和工作方式,然后必须将启动命令写入命令寄存器。即使定时/计数器已经计数,在写入启动命令后仍可改变定时器的工作方式。

## 5. AT89S52 单片机扩展 8155 的接口应用

### (1) AT89S52 与 8155 的基本连接方法

AT89S52 单片机扩展 8155 后,直接为系统增加了 256 B 片外 RAM,22 位 I/O 口线及一个 14 位定时器。具体电路如图 10.4.8 所示。

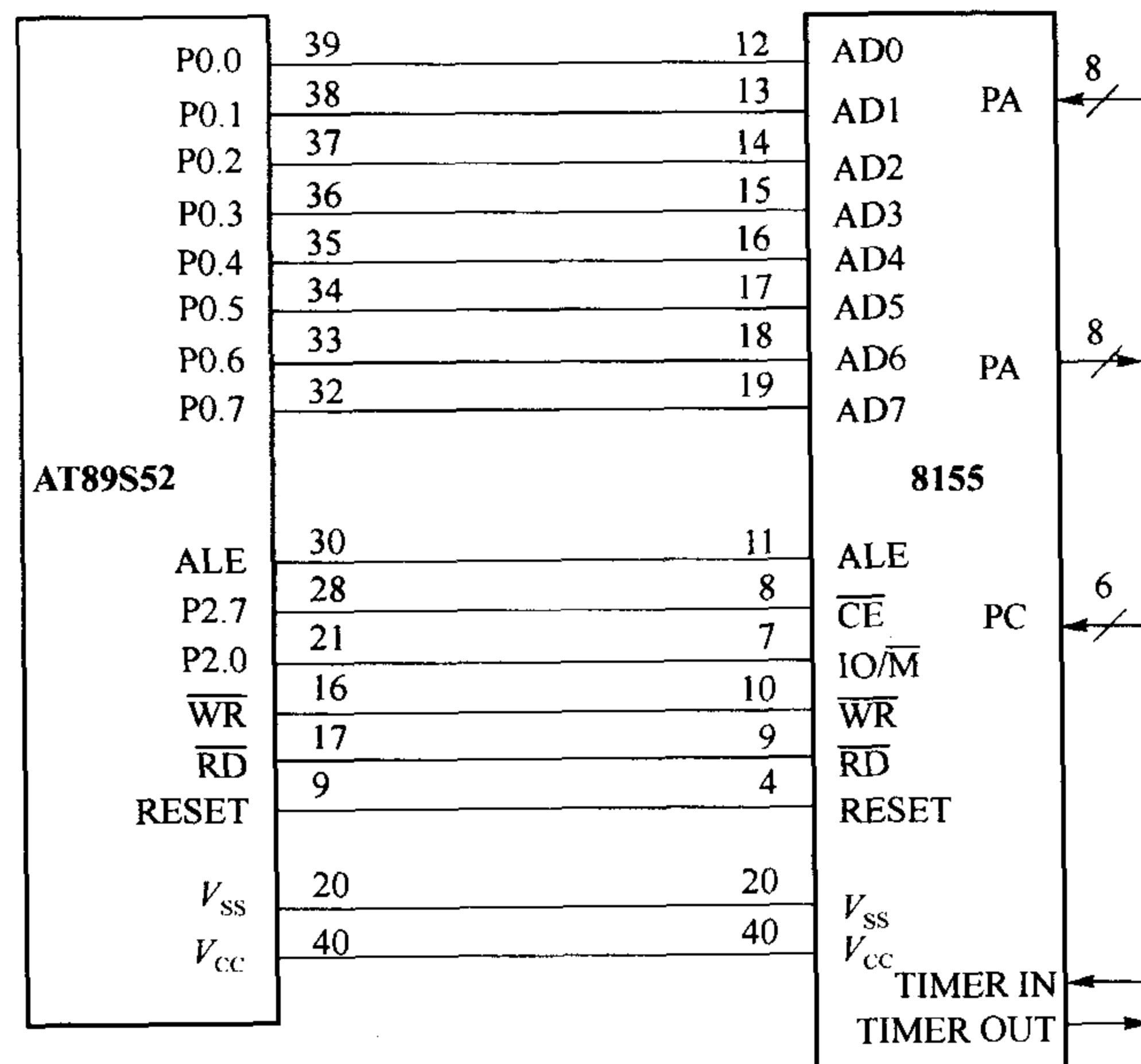


图 10.4.8 AT89S52 与 8155 的连接方法

由于 8155 芯片内部有地址锁存器,因此单片机 P0 口输出的低 8 位地址不需另加锁

存器,可直接与 8155 的 AD0~AD7 相连,这样单片机 P0 口既作低 8 位地址总线,又作数据总线,地址用 ALE 信号锁存到 8155 中。高 8 位地址由 $\overline{CE}$ 和 $\overline{IO}/\overline{M}$ 的地址控制线决定,所以图中连接状态下的地址如表 10.4.2 所示。

表 10.4.2 8155 中 RAM 和 I/O 口地址分配表

RAM 地 址		7E00H~7EFFH
I/O 接 口 地 址	命令/状态寄存器	7F00H
	PA 口	7F01H
	PB 口	7F02H
	PC 口	7F03H
	定时/计数器低 8 位寄存器	7F04H
	定时/计数器高 8 位寄存器	7F05H

**例 10-2** 对于图 10.4.8 的电路,编写程序向 8155 中 RAM 的 6EH 单元送入立即数 17H。

**解** 其汇编语言程序清单如下(只有通过累加器 A 才能把数据送到外部 RAM 中)。

```
MOV    A, #17H           ;立即数 17 送累加器 A
MOV    DPTR, #7E6EH      ;指向 8155 的 6EH 单元
MOVX   @DPTR, A          ;把立即数 17H 送入 8155 的 6EH 单元
```

**例 10-3** 编写程序,将 8155 设置为 I/O 接口和定时器工作方式, A 口定义为基本输入方式, B 口定义为基本输出方式, 定时器作为方波发生器。对输入脉冲进行 24 分频(8155 中定时器最高计数频率为 4 MHz)。

**解** 8155 的操作程序如下。

```
MOV    DPTR, #7F04H      ;指向定时器低 8 位
MOV    A, #18H           ;计数常数 0018H = 24
MOVX   @DPTR, A          ;装入计数常数低 8 位
INC    DPTR              ;指向定时器高 8 位
MOV    A, #40H           ;设置定时器方式为连续方波输出
MOVX   @DPTR, A          ;装入定时器高 8 位
MOVX   DPTR, #7F00H      ;指向命令/状态寄存器
MOV    A, #C2H           ;命令控制字设定为 A 口基本输入方式, B 口基
                        ;本输出方式;并启动定时器
MOVX   @DPTR, A          ;命令字送命令寄存器,启动计数
```

## (2) 利用 8155 芯片扩展键盘/显示器接口

### 1) 硬件电路

图 10.4.9 是用 8155 实现的 AT89S52 单片机与 4×8 键盘、8 位 LED 显示器(8 位 8 段共阴极)的接口电路。图 10.4.9 中 AT89S52 外扩一片 8155, 8155 片内 RAM 地址为 7E00H~7EFFH, 8155 的 I/O 接口地址为 7F00H~7F05H。8155 的端口 A 为输出口, 作为键盘扫描口, 控制键盘的列线, 同时又是 LED 显示器扫描口, 控制每位显示器的公共阴极的电位。8155 的端口 B 作为显示器的段数据口; 端口 C 作为输入口, 口线 PC0~

PC3 分别接键盘行线,称为键盘输入口。LED 的段、位信号均由 8 位集电极开路输出的 8718 驱动器驱动。

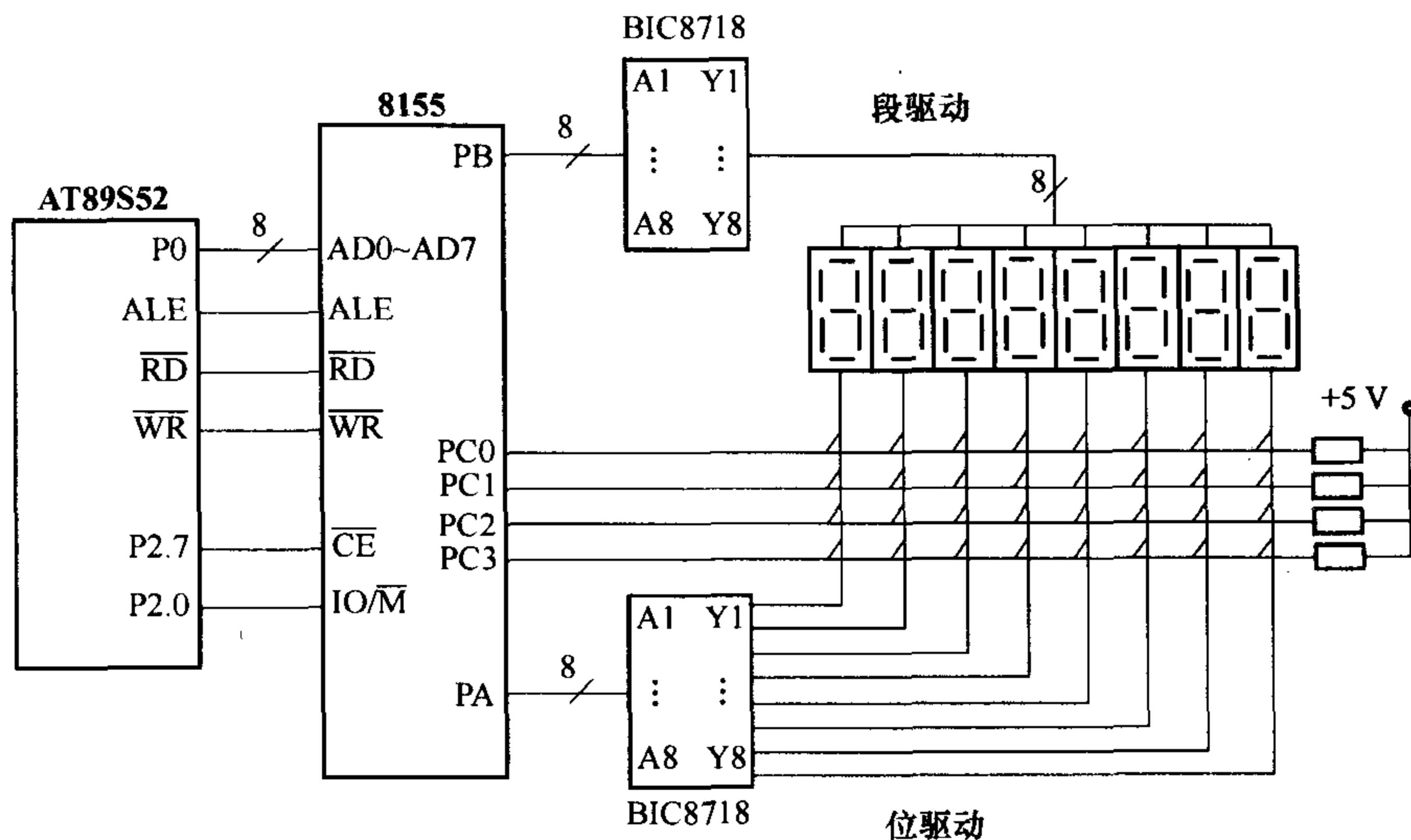


图 10.4.9 8155 实现键盘/显示器原理框图

LED 采用软件译码动态扫描显示工作方式,键盘采用逐列扫描查询工作方式。

## 2) 软件设计

软件包括动态显示子程序和键盘扫描子程序。具体介绍如下。

### ① 动态显示子程序

对图 10.4.9 中的 8 位 LED 显示器,在 AT89S52 内部 RAM 中设置 8 个显示缓冲单元,分别存放显示的 8 位数据。8155 的 A 口扫描输出总是只有一位为低电平,即显示器的 8 位中仅有一位公共阴极为低电平,其他位公共阴极均为高电平;8155 的 B 口输出阴极为低的相应位要显示的数据的段选码,使该位显示出一个字符,其他位为暗。依次改变 A 口输出为高电平的位,B 口输出对应的段选码,显示器就显示由缓冲区中的显示数据所确定的字符。动态显示程序的流程图如图 10.4.10 所示。

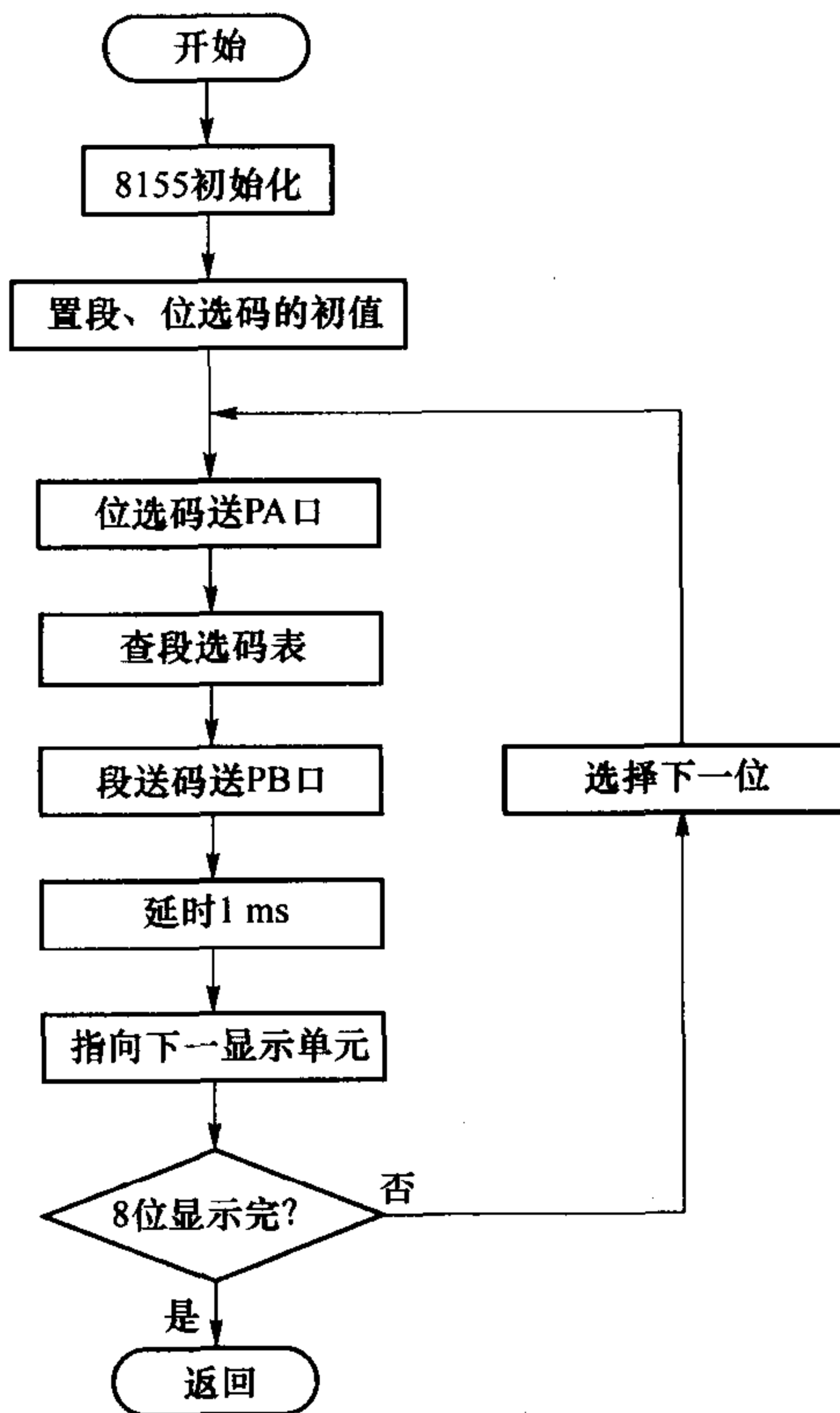


图 10.4.10 动态显示子程序流程图



程序清单如下:

```

DIS:  MOV    A, #00000011b           ;初始化 8155 PA、PB 口为输出;
                                           ;PC 口为输入
        MOV    DPTR, #7F00H          ;8155 命令口地址送 DPTR
        MOVX   @DPTR, A              ;写命令
        MOV    R0, #70H              ;70H~77H 单元存 8 个显示数据
        MOV    R3, #7FH              ;第一位 LED 的位选码 7FH
        MOV    A, R3
AGAIN:  MOV    DPTR, #7F01H          ;指向 PA 口
        MOVX   @DPTR, A              ;位选码送 PA 口
        MOV    A, @R0                ;取显示数据
        MOV    DPTR, #DSEG           ;取段选码表首址
        MOVC   A, @A + DPTR          ;取段选码
        MOV    DPTR, #7F02H          ;指向 PB 口
        MOVX   @DPTR, A              ;段选码送 PB 口
        ACALL  DL1ms                 ;延时 1 ms
        INC    R0                    ;指向下一显示数据单元
        MOV    A, R3
        JNB    ACC.0, OUT            ;8 位显示完, 转 OUT
        RR     A                      ;未完, 调整为下一位位选码
        MOV    R3, A
        AJMP   AGAIN                ;继续显示下一位
OUT:    RET                          ;子程序返回
DSEG:   DB     3FH, 06H, 5BH, 4FH, 66H, 6DH ;显示 0, 1, 2, 3, 4, 5
        DB     7DH, 07H, 7FH, 6FH, 77H, 7CH ;显示 6, 7, 8, 9, A, B
        DB     39H, 5EH, 79H, 71H          ;显示 C, D, E, F
DL1ms:  MOV    R7, #01H              ;延时 1 ms 子程序
DL0:    MOV    R6, #0FFH
DL1:    DJNZ   R6, DL1
        DJNZ   R7, DL0
        RET

```

## ② 键盘扫描子程序

键盘扫描程序应有以下 4 个方面的功能。第一, 判别键盘上有无键闭合。其方法是使扫描口 A 的 PA0~PA7 输出全 0, 读端口 C 的状态, 若 PC0~PC3 全为 1, 即键盘行线全为高电平, 则键盘上没有闭合键; 若 PC0~PC3 不全为 1, 则有键处于闭合状态。第二, 去除键的机械抖动。其方法为判别出键盘上有闭合键后, 延时一段时间再判别键盘的状态, 若仍有键闭合, 则认为键盘上有一个键处于稳定的闭合期, 否则认为是键的抖动。第三, 判别闭合键的键号。其方法为对键盘的列线进行扫描, 扫描口 A 输出数据, 每次输出的数

据仅有一位为低电平,其余均为高电平。相应地依次读出 C 口的状态,若 PC0~PC3 全为 1,则列线为 0 的这一列上没有键闭合;否则闭合键的键号等于为低电平的列号加上为低电平的行的首键号。例如,A 口输出 PA0~PA7 为 10111111 时,读入 PC0~PC3 为 1011,则 1 行 1 列相交的键处于闭合状态,第一行的首键号为 8 列号为 1,则闭合键号为  $N$ , $N = \text{行首键号} + \text{列号} = 8 + 1 = 9$ 。第四,对键的一次闭合仅做一次处理。其方法为等待闭合键释放后再进行处理。

在扫描键盘的过程中应兼顾显示器的显示,键盘扫描子程序框图如图 10.4.11 所示。

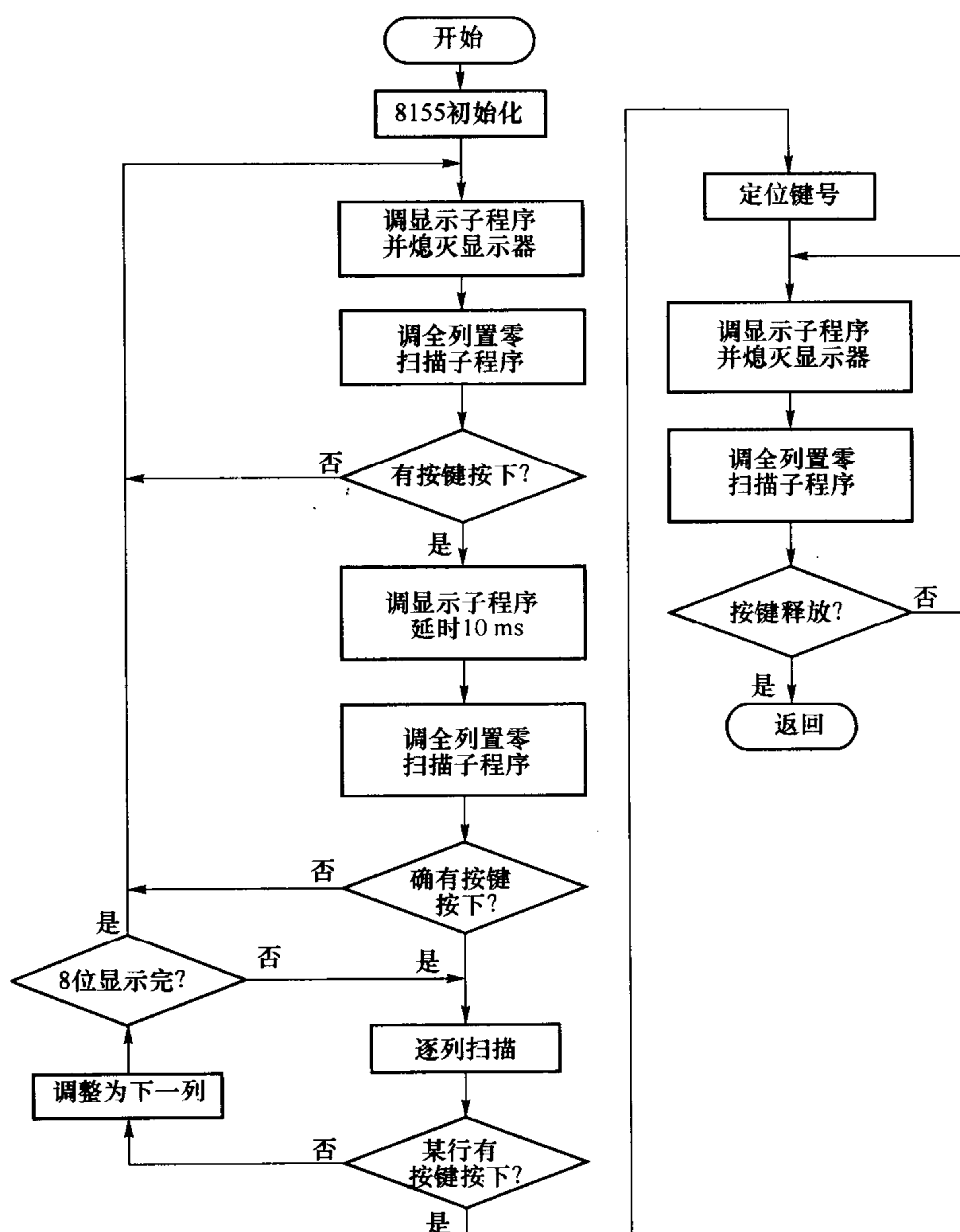


图 10.4.11 键盘扫描子程序框图

程序清单如下:

```

KEYSUB:  MOV    A, #00000011B    ;8155 初始化 PA 口, PB 口输出, PC 口输入
          MOV    DPTR, #7F00H    ;控制字地址送 DPTR
  
```

	MOVX	@DPTR,A	;向 8155 输出控制字
BEGIN:	ACALL	DIS	;调显示子程序
	ACALL	CLEAR	;清零显示器,即熄灭
	ACALL	CCSCAN	;全列置零扫描,判有无按键按下
	JNZ	INK1	;有键按下,转 INK1
	AJMP	BEGIN	
INK1:	ACALL	DIS	;调显示子程序,延时 8~9 ms
	ACALL	DL1ms	
	ACALL	DL1ms	;共延时约 10 ms 去抖
	ACALL	CLEAR	;熄灭显示器
	ACALL	CCSCAN	;全列置零扫描,判是否确有按键按下
	JNZ	INK2	;确有键按下,转 INK2
	AJMP	BEGIN	;抖动引起,转回 BEGIN
INK2:	MOV	R2,#0FEH	;扫描第一列,置第一列为 0
	MOV	R4,#00H	;列号送 R4
COLUM:	MOV	DPTR,#7F01H	;指向 PA 口
	MOV	A,R2	;扫描码送 A
	MOVX	@DPTR,A	;输出扫描码
	INC	DPTR	
	INC	DPTR	;指向 PC 口
	MOVX	A,@DPTR	;读入 PC 口
	JB	ACC.0,LONE	;第一行无按键按下,转 LONE
	MOV	A,#00H	;第一行有按键按下,行码送 A
	AJMP	KCODE	;转 KCODE,定按键的键号
LONE:	JB	ACC.1,LTWO	;第二行无按键按下,转 LTWO
	MOV	A,#08H	;第二行有按键按下,行码送 A
	AJMP	KCODE	
LTWO:	JB	ACC.2,LTHR	;第三行无按键按下,转 LTHR
	MOV	A,#10H	;第三行有按键,行码送 A
	AJMP	KCODE	
LTHR:	JB	ACC.3,NEXT	;第四行无按键按下,转扫描下一列
KCODE:	ADD	A,R4	;行首键号加列号得键号
	PUSH	A	;键号入栈保护
KON:	ACALL	DIS	;调显示,等待按键释放
	ACALL	CLEAR	;熄灭显示
	ACALL	CCSCAN	;判按键是否仍按下
	JNZ	KON	;键未释放,继续等待
	POP	A	;恢复键号到 A 中

```

                RET                ; 返回
NEXT:          INC      R4        ; 列号加 1
                MOV      A, R2    ; 列扫描码送 A
                JNB      ACC. 7, KERR ; 全 8 列扫描完, 无按键, 为干扰, 转 KERR
                RL       A        ; 调整为下一列扫描码
                MOV      R2, A    ; 保存扫描码
                AJMP     COLUMN    ; 继续扫描下一列
KERR:          AJMP     BEGIN    ; 继续等待键输入
CCSCAN        MOV      DPTR, #7F01H ; 指向 PA 口
                MOV      A, #00H
                MOVX     @DPTR, A ; PA 口输出全零
                INC      DPTR
                INC      DPTR    ; 指向 PC 口
                MOVX     A, @DPTR ; 读 PC 口
                CPL      A
                ANL      A, #0FH ; 屏蔽高 4 位
                RET                ; 返回
CLEAR:        MOV      DPTR, #7F02H ; 指向 PB 口
                MOV      A, #00H ; 段选码为 00H
                MOVX     @DPTR, A ; PB 口输出全 0
                RET                ; 返回

```

### 10.4.3 8255A 可编程并行 I/O 接口扩展

8255A 是 Intel 公司生产的可编程输入输出接口芯片, 具有 3 个 8 位的并行 I/O 接口, 3 种工作方式, 可通过程序改变其功能, 可以与单片机直接相连, 因而使用灵活方便, 通用性好, 可作为单片机与多种外围设备连接时的中间接口电路。8255A 的引脚及内部结构如图 10.4.12 所示。

#### 1. 8255A 的引脚说明

8255A 共有 40 个引脚, 双列直插式封装的引脚功能如下。

D7~D0: 三态双向数据线, 与单片机数据总线连接, 用来传送数据信息。

$\overline{CS}$ : 片选信号线, 低电平有效, 表示芯片被选中。

$\overline{RD}$ : 读出信号线, 低电平有效, 控制数据的读出。

$\overline{WR}$ : 写入信号线, 低电平有效, 控制数据的写入。

PA0~PA7: 端口 A 的输入/输出线。

PB0~PB7: 端口 B 的输入/输出线。

PC0~PC7: 端口 C 的输入/输出线。

RESET: 复位信号线。若 RESET=1, 8255 复位。复位状态控制寄存器被清除, A, B, C 口被置为输入方式。

A1~A0:地址线,用来选择 8255A 内部端口。

## 2. 内部结构

8255A 的内部结构如图 10.4.12 所示,其中包括 3 个并行数据输入/输出端口,2 个工作方式控制电路,一个读/写控制逻辑电路和 8 位总线缓冲器。各部分功能概括如下。

### (1) 端口 A、B、C

8255A 有 3 个 8 位并行口,即 A、B 和 C,都可以选择作为输入或输出工作方式,每个并行口都具有一个 8 位数据输出锁存器/缓冲器,一个 8 位数据输入锁存器。但在功能和结构上有些差异。

A 口、B 口作为数据输入/输出端口,C 口既可作为输入/输出端口使用,又可在软件的控制下,分为两个 4 位的端口,作为 A 口、B 口选通方式操作时的控制信号。

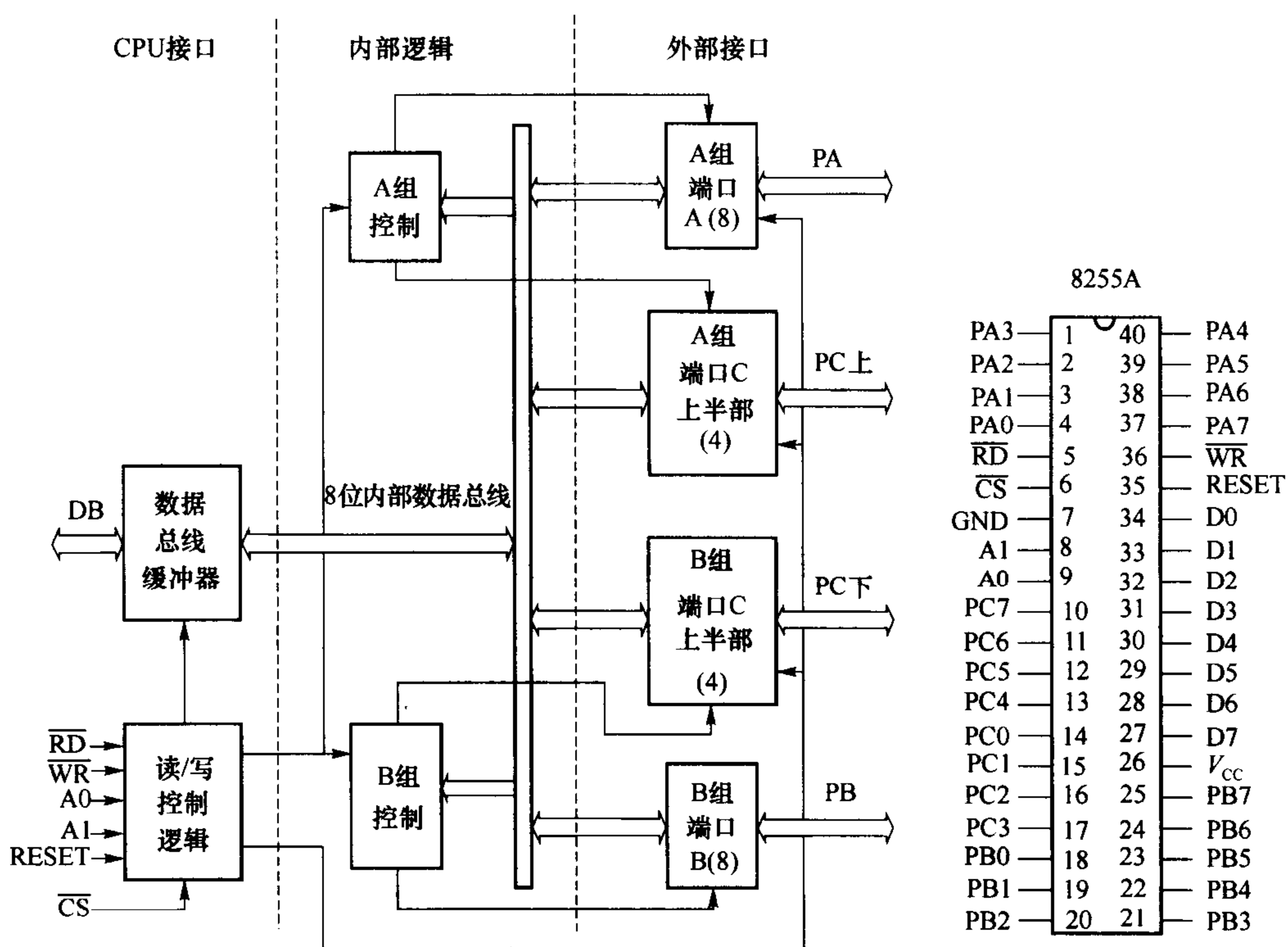


图 10.4.12 8255A 的引脚图及内部结构框图

### (2) 工作方式控制电路

工作方式控制电路包括 A 组控制电路和 B 组控制电路。两组控制电路共用一个控制命令寄存器,用来接收 CPU 发来的控制字,以决定两组端口的工作方式,也可根据控制字的要求对 C 口按位清 0 或者按位置 1。

A 组控制电路用来控制 A 口和 C 口的上半部分(PC7~PC4)。B 组控制电路用来控制 B 口和 C 口的下半部分(PC3~PC0)。

### (3) 数据总线缓冲器

数据总线缓冲器是一个三态双向 8 位缓冲器,作为 8255A 与系统总线之间的接口,用来传送数据、指令、控制命令以及外部状态信息。

### (4) 读/写控制逻辑电路

读/写控制逻辑电路接收 CPU 发来的控制信号  $\overline{RD}$ ,  $\overline{WR}$ , RESET 以及地址信号 A1 ~ A0 等,然后根据控制信号的要求,将端口数据读出,送到 CPU;或者将 CPU 送来的数据写入端口。各端口的工作状态如表 10.4.3 所示。

表 10.4.3 8255A 端口的操作状态

端口地址选择				操作选择		CPU 操作功能
$\overline{CS}$	A1	A0	所选端口	$\overline{RD}$	$\overline{WR}$	
0(选中)	0	0	A 口	0	1	读 A 口内容
	0	1	B 口	0	1	读 B 口内容
	1	0	C 口	0	1	读 C 口内容
	0	0	A 口	1	0	写入 A 口
	0	1	B 口	1	0	写入 B 口
	1	0	C 口	1	0	写入 C 口
	1	1	控制寄存器	1	0	写入控制字
1	×	×	未选中	×	×	D0~D7 三态

## 3. 8255A 的工作方式

8255A 有 3 种基本工作方式。方式 0:基本输入输出。方式 1:选通输入输出。方式 2:双向传送。

### (1) 工作方式 0

基本输入/输出方式。这种方式不需选通信号。PA, PB 和 PC 中任一端口都可以通过方式控制字设定为输入或输出。

### (2) 工作方式 1

工作方式 1 是选通输入/输出工作方式。该方式时 3 个端口分为两组,即 A 组和 B 组。每一组包括一个 8 位数据端口和一个 4 位的控制/状态端口。每一个 8 位数据端口均可设置为输入或者输出,输入输出均可锁存。4 位端口作为 8 位数据端口的控制/状态信号端口。A 组包括 A 口和 PC7~PC4, A 口可以由编程设定为输入或输出, PC7~PC4 作为输入/输出操作的选通信号和应答信号。B 组包括 B 口和 PC3~PC0,用法同 A 组。

### (3) 工作方式 2

只有 A 口可以工作于方式 2。按照方式 2 工作时, A 口成为双向数据总线端口,既可以发送数据,又可以接收数据。此时 C 口有 5 条线被固定为 A 口与外设之间的联络信号线。C 口余下的 3 条线可以作为 B 口方式 1 下的联络线;也可以和 B 口一起作为方式 0 的 I/O 口线。

这 3 种工作方式如图 10.4.13 所示。它们由 CPU 向控制端口输出的控制字来选择。

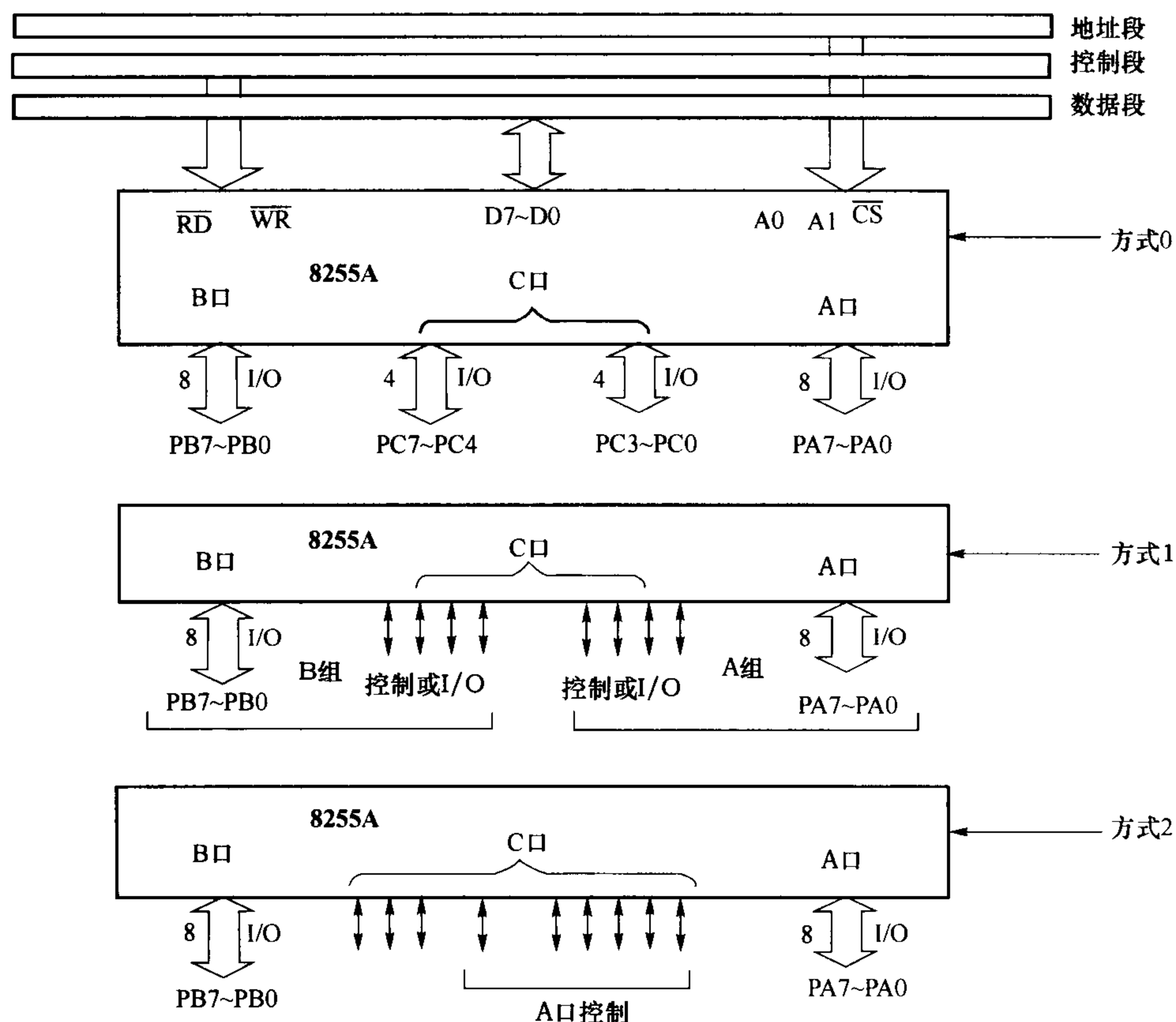


图 10.4.13 8255A 的工作方式示意图

各引脚的功能如表 10.4.4 所示。

表 10.4.4 8255A 的 C 口联络控制信号线

C 口的位	方式 1(A 口、B 口)		方式 2(仅用于 A 口)	
	输入	输出	输入	输出
PC0	INTRB	INTRB	I/O	I/O
PC1	IBFB	$\overline{\text{OBF}}\text{B}$	I/O	I/O
PC2	$\overline{\text{STB}}\text{B}$	$\overline{\text{ACK}}\text{B}$	I/O	I/O
PC3	INTRA	INTRA	INTRA	INTRA
PC4	$\overline{\text{STB}}\text{A}$	I/O	$\overline{\text{STB}}\text{A}$	×
PC5	IBFA	I/O	IBFA	×
PC6	I/O	$\overline{\text{ACK}}\text{A}$	×	$\overline{\text{ACK}}\text{A}$
PC7	I/O	$\overline{\text{OBF}}\text{A}$	×	$\overline{\text{OBF}}\text{A}$

表中 I/O 表示 C 口未用的这些线可以设定为一般的输入/输出线。表中各联络线用于输入时定义如下：



- ①  $\overline{\text{STB}}$ (Strobe), 选通信号输入线, 低电平有效。
- ② IBF(Input Buffer Full), 输入缓冲器满, 高电平有效。
- ③ INTR(Interrupt Request), 中断请求信号, 高电平有效。

在 $\overline{\text{STB}}=\text{IBF}=1$  时, 8255A 会向 CPU 发出中断请求信号  $\text{INTR}=1$ 。在 CPU 响应中断后读取缓冲器的数据时, 由 $\overline{\text{RD}}$ 的下降沿将 INTR 清 0, 通知外设再一次输入数据。

表中用于输出的联络信号定义如下:

①  $\overline{\text{ACK}}$ (Acknowledge), 外设响应输入信号, 低电平有效。当 $\overline{\text{ACK}}=0$  时, 表示外设已取走并且处理完 CPU 通过 8255A 输出的数据。

②  $\overline{\text{OBF}}$ (Output Buffer Full), 输出缓冲器满, 低电平有效。当 $\overline{\text{OBF}}=0$  时, 表示 CPU 已经把数据写入 8255A 指定的端口, 通知外设可以把数据取走。

③ INTR, 中断请求输出信号, 高电平有效。当  $\text{INTR}=1$  时, 向 CPU 申请中断, 要求 CPU 继续输出数据, CPU 在中断程序中把数据写入 8255A。

#### 4. 8255A 的控制字

8255A 共有两个控制字, 用来选择工作方式或对 C 口控制。

##### (1) 方式控制字

8255A 的 3 个端口工作在什么方式, 是输入还是输出, 都是由工作方式控制字设定, 方式控制字的格式如图 10. 4. 14 所示。

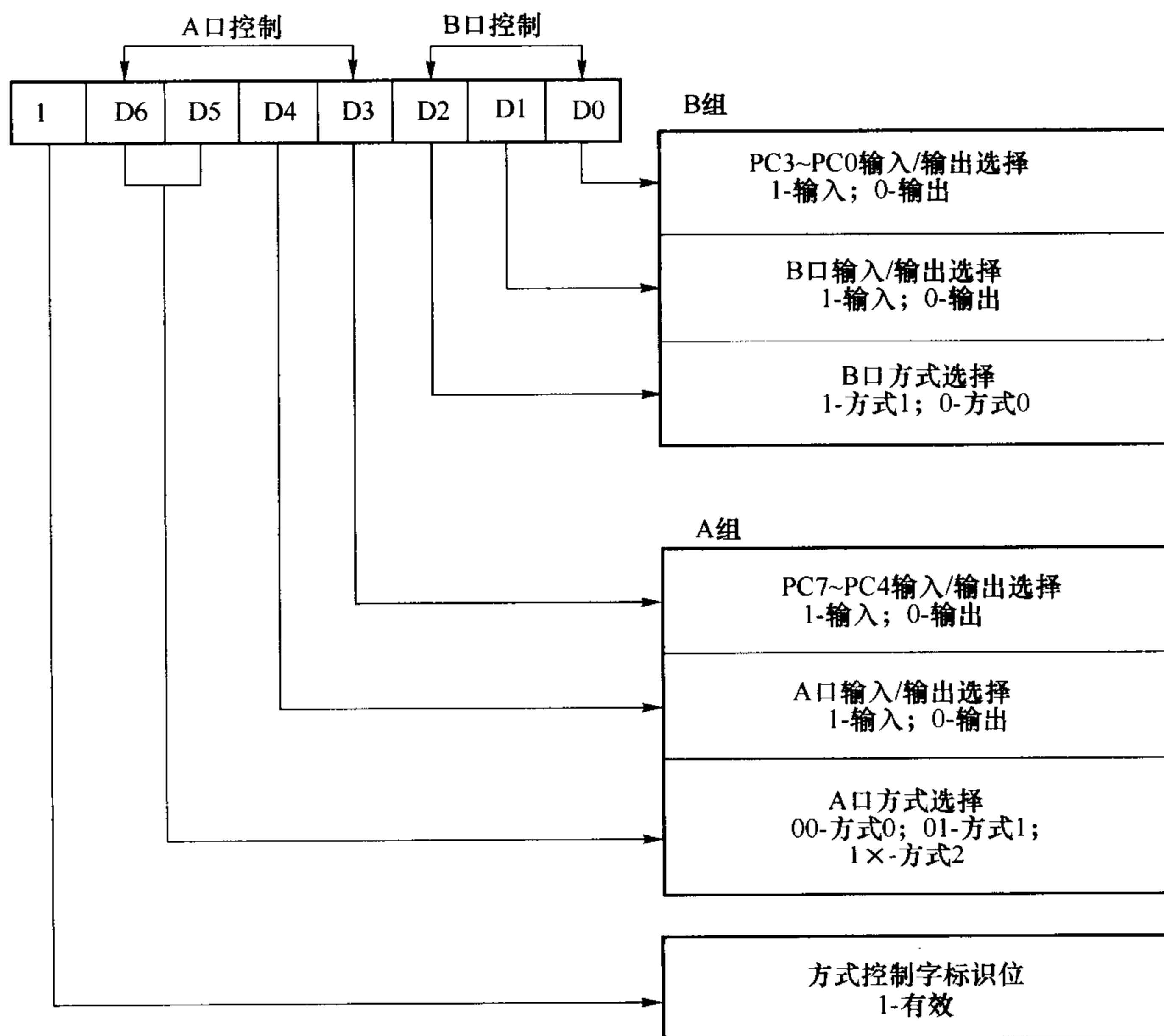


图 10. 4. 14 8255A 方式选择控制字

另外,在用户选择方式 1 或方式 2 时,对 C 口的定义无论是输入还是输出方式,都不影响 C 口作为控制联络线使用的各位功能,但未用于控制联络线的各位,仍用 D0,D3 定义。

8255A 作为 I/O 应用时,首先应对其进行初始化,设置 A,B,C 口的工作方式,然后才可以对 A,B,C 口进行读写操作。否则按照复位状态进行操作。

**例 10-4** 设 A 口地址为 FF7CH;B 口地址为 FF7DH;C 口地址为 FF7EH;控制字地址为 FF7FH。要求 8255A 工作在方式 0,A 口作为输入,B 口、C 口作为输出。编写相应程序。

**解** 8255A 相应的控制字为 10010000B=90H

初始化程序如下。

```
MOV    A, #90H           ;方式 0,A 口输入,B 口、C 口输出
MOV    DPTR, #0FF7FH     ;控制寄存器地址送 DPTR
MOVX   @DPTR, A          ;方式控制字送控制寄存器
```

若读 A 口数据:

```
MOV    DPTR, #0FF7CH     ;A 口地址送 DPTR
MOVX   A, @DPTR          ;从 A 口读数据
```

若把 DATA1 数据写 B 口:

```
MOV    DPTR, #0FF7DH     ;送 B 口地址到 DPTR
MOV    A, #DATA1          ;将 DATA1 送累加器 A
MOVX   @DPTR, A          ;将 DATA1 送 B 口输出
```

## (2) 端口 C 置位/复位控制字

由于 C 口常作为联络控制位使用,应使 C 口各位用置位/复位控制字单独设置,以实现用户要求的控制功能,格式如图 10.4.15 所示。

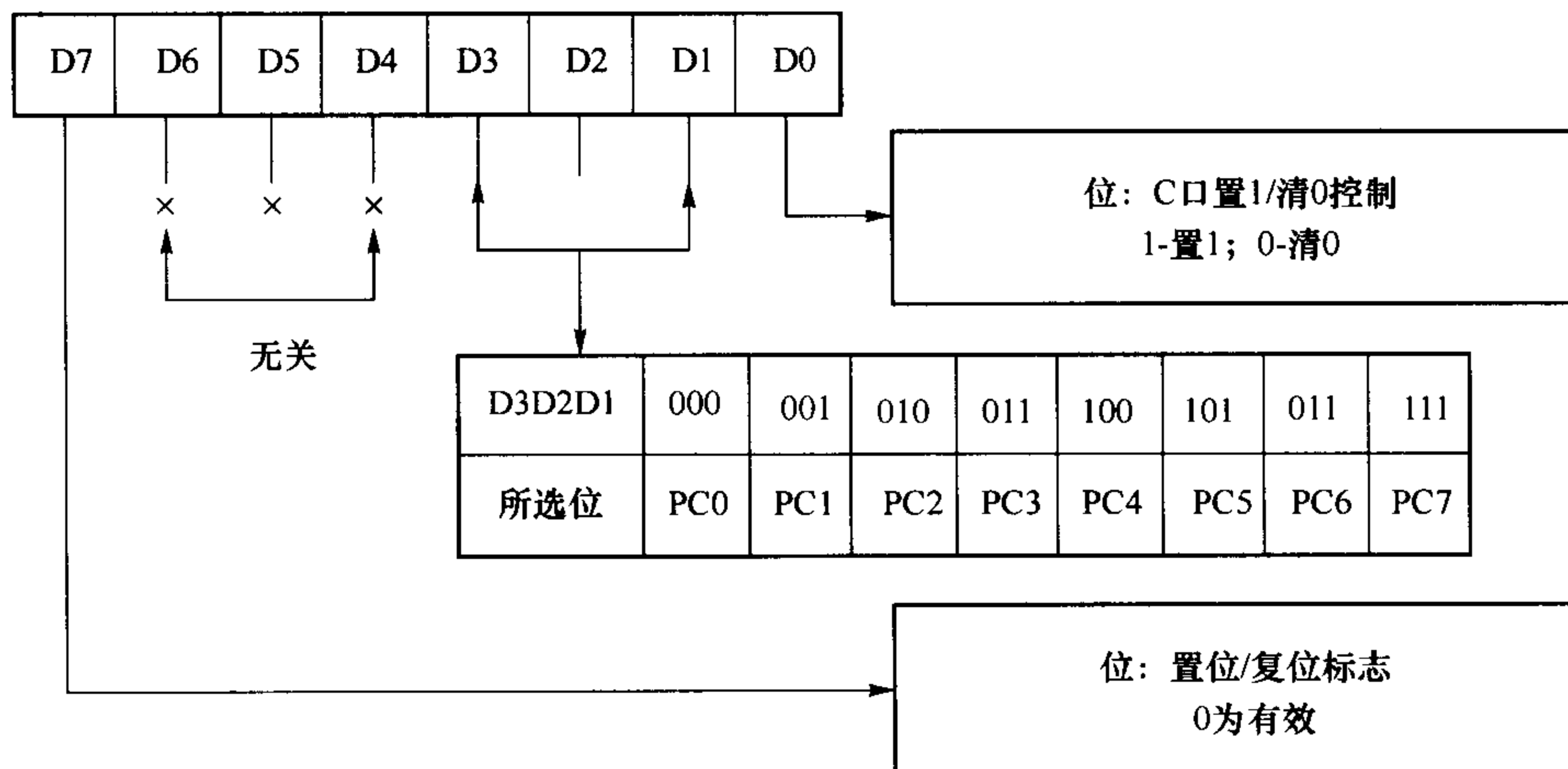


图 10.4.15 端口 C 按位置位/复位控制字

由图 10.4.15 可见,对 C 口的某一位 PC<sub>i</sub> 进行置位或复位操作由 D0 设定,选择 C 口的哪一位进行操作是由 D3,D2,D1 来确定。

D7=0 是置位/复位控制字的特征位,由于 8255A 中两个控制寄存器共用一个端口

地址,故 8255A 根据控制字的 D7 位来区别是方式控制字,还是 C 口置位/复位控制字(比较图 10.4.14 中的 D7)。

**例 10-5** 设控制字地址为 FF7FH,分别置位和复位 C 口的第 5 位,即 PC4。编写相应程序。

**解** 置位 PC4 的控制字为 00001001B=9H。

置位程序如下:

```
MOV    DPTR, #0FF7FH    ;送控制字地址到 DPTR
MOV    A, #9H           ;将控制字送累加器 A
MOVX   @DPTR, A         ;将控制字送控制寄存器
                        ;执行 PC4 = 1
```

复位 PC4 的控制字为 00001000B=8H。

复位程序如下:

```
MOV    DPTR, #0FF7FH    ;送控制字地址到 DPTR
MOV    A, #8H           ;将控制字送累加器 A
MOVX   @DPTR, A         ;将控制字送控制寄存器
                        ;执行 PC4 = 0
```

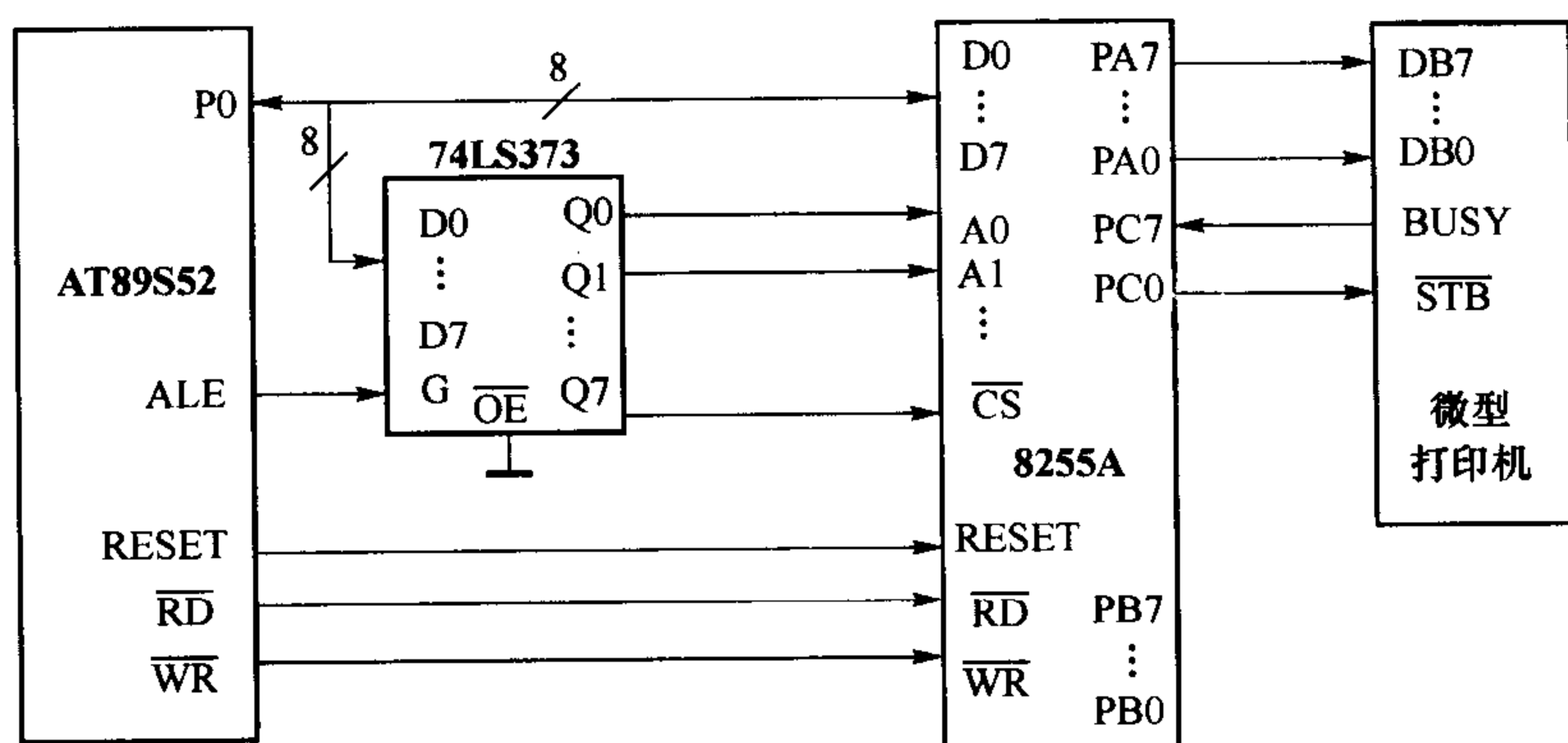


图 10.4.16 AT89S52 通过扩展 8255A 与微型打印机接口

**例 10-6** 图 10.4.16 中,8255A 的片选线与 P0.7 连接,打印机与 AT89S52 采用查询方式交换数据。打印机的状态信号 BUSY 输入给 PC7,打印机忙时 BUSY=1。打印机的数据输入采用选通控制,当  $\overline{\text{STB}}$  上出现下降沿时数据被写入,要求编写向打印机输出 80 个数据的程序。

**解** 8255A 的 A, B, C 和控制寄存器的口地址分别为 7CH, 7DH, 7EH 和 7FH。8255A 采用方式 0,由 PC0 模拟产生  $\overline{\text{STB}}$  信号。因 PC7 输入,PC0 输出,则方式选择控制字为 10001110B=8EH。自内部 RAM 20H 单元开始向打印机输出 80 个数据的程序如下。

```
MOV    R0, #7FH        ;R0 指向控制口
MOV    A, #8EH         ;方式控制字为 8EH
MOVX   @R0, A          ;送方式控制字
```

MOV	R1, #20H	;送内部 RAM 数据块首地址
MOV	R2, #50H	;置数据块长度
LP: MOV	R0, #7EH	;R0 指向 C 口
LP1: MOVX	A, @R0	;读 PC7 即 BUSY 状态
JB	ACC. 7, LP1	;BUSY = 1, 查询等待打印机
MOV	R0, #7CH	;指向 A 口
MOV	A, @R1	;取 RAM 数据
MOVX	@R0, A	;数据输出到 8255A 口锁存
INC	R1	;RAM 地址加 1
MOV	R0, #7FH	;指向控制口
MOV	A, #00H	;PC0 复位控制字
MOVX	@R0, A	;PC0 = 0, 产生 $\overline{\text{STB}}$ 的下降沿
MOV	A, #01H	;PC0 置位控制字
MOVX	@R0, A	;PC0 = 1, 产生 $\overline{\text{STB}}$ 的上升沿
DJNZ	R2, LP	;未完, 则重复

## 10.5 A/D 和 D/A 转换接口的扩展

单片机本身处理的是数字量,然而在单片机的测控系统中,常检测到的是连续变化的模拟量。这些量(如温度、压力、流量和速度等)只有被转换成离散的数字量后,才能输入到单片机中进行处理;然后再将处理结果的数字量经反变换,变成模拟量,实现对被控对象(过程、仪表、机电设备、装置)的控制。这时就需要解决单片机与 A/D 和 D/A 的接口问题。

A/D 转换器用以实现模拟量向数字量的转换。按转换原理可分为:计数式、双积分式、逐次逼近式 A/D 转换器。逐次逼近式 A/D 转换器是一种转换速度较快,精度较高,价格适中的转换器。其转换时间大约在几微秒到几百微秒之间。

A/D 转换器的主要技术指标如下。

### (1) 分辨率

分辨率是指 A/D 转换器能够分辨出来的输出模拟电压的最小变化量,反映了对输入模拟信号的最小变化的分辨能力,由下式计算。

$$\Delta = \frac{\text{满量程输入电压}}{2^n - 1}$$

其中,  $n$  为 A/D 转换器的位数。

### (2) 量化误差

量化误差是指由 A/D 转换器有限的分辨率而引起的误差。量化误差有两种表示方法,一种是绝对量化误差;另一种是相对量化误差。

绝对量化误差  $\epsilon = \frac{\Delta}{2}$

相对量化误差

$$\epsilon = \frac{1}{2^{n+1}}$$

### 10.5.1 8 位并行 A/D 转换器 ADC0809 的扩展

#### 1. ADC0809 的结构原理

ADC0809 是美国国家半导体公司生产的 8 位 ADC, 采用逐次逼近的方法完成 A/D 转换。ADC0809 的内部结构如图 10.5.1 所示。ADC0809 由单一 +5 V 电源供电, 片内有带锁存功能的 8 路模拟多路开关, 可对 8 路 0~5 V 的输入模拟电压信号分时进行转换, 完成一次转换约需 100  $\mu$ s; 输出具有 TTL 三态锁存缓冲器, 可直接接到单片机数据总线上。通过适当的外接电路, ADC0809 可对 0~5 V 的双极性模拟信号进行转换。

#### 2. ADC0809 的引脚及功能

ADC0809 是 28 脚双列直插式封装, 引脚如图 10.5.2 所示。各引脚功能说明如下。

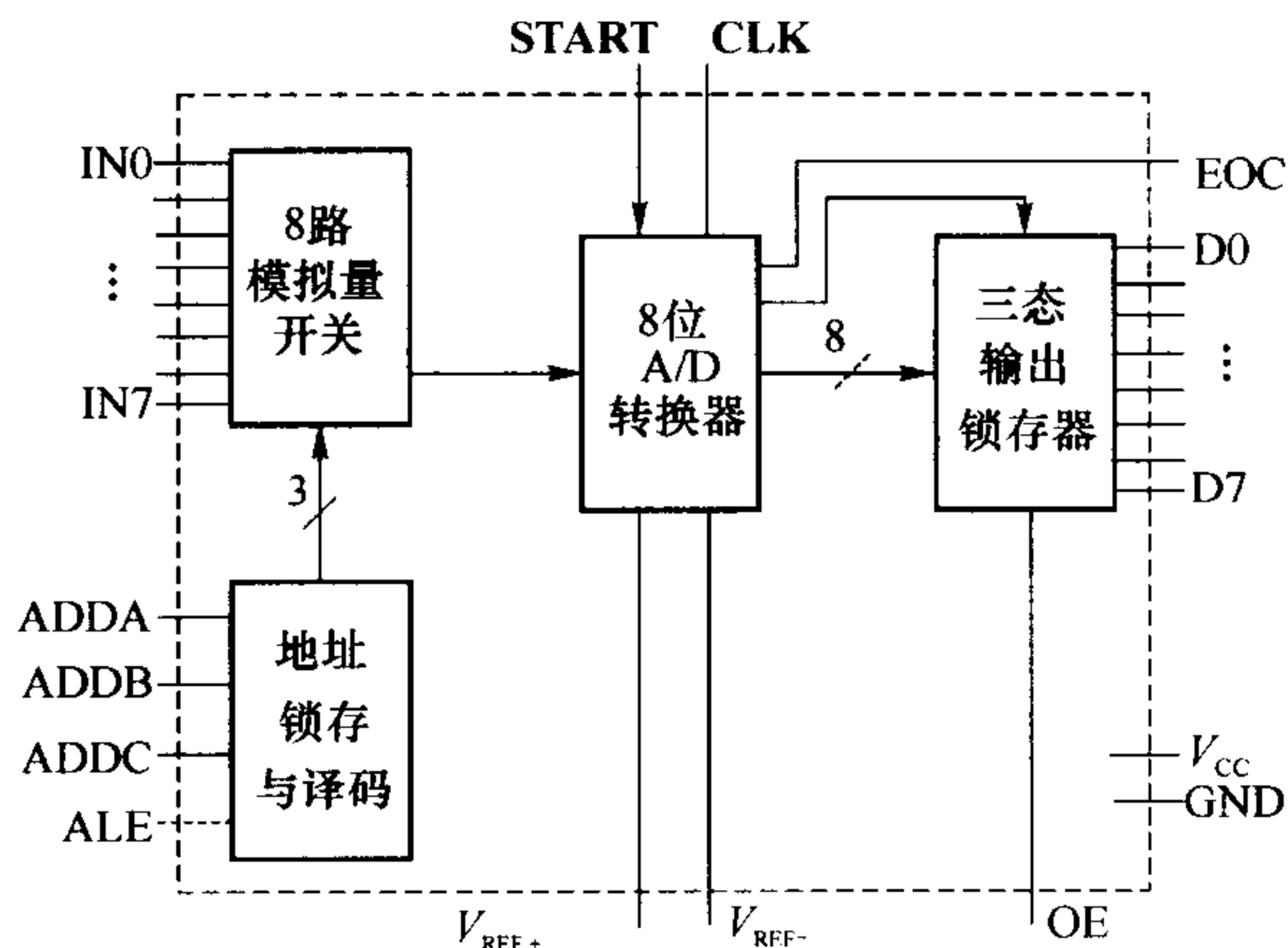


图 10.5.1 ADC0809 的内部结构框图

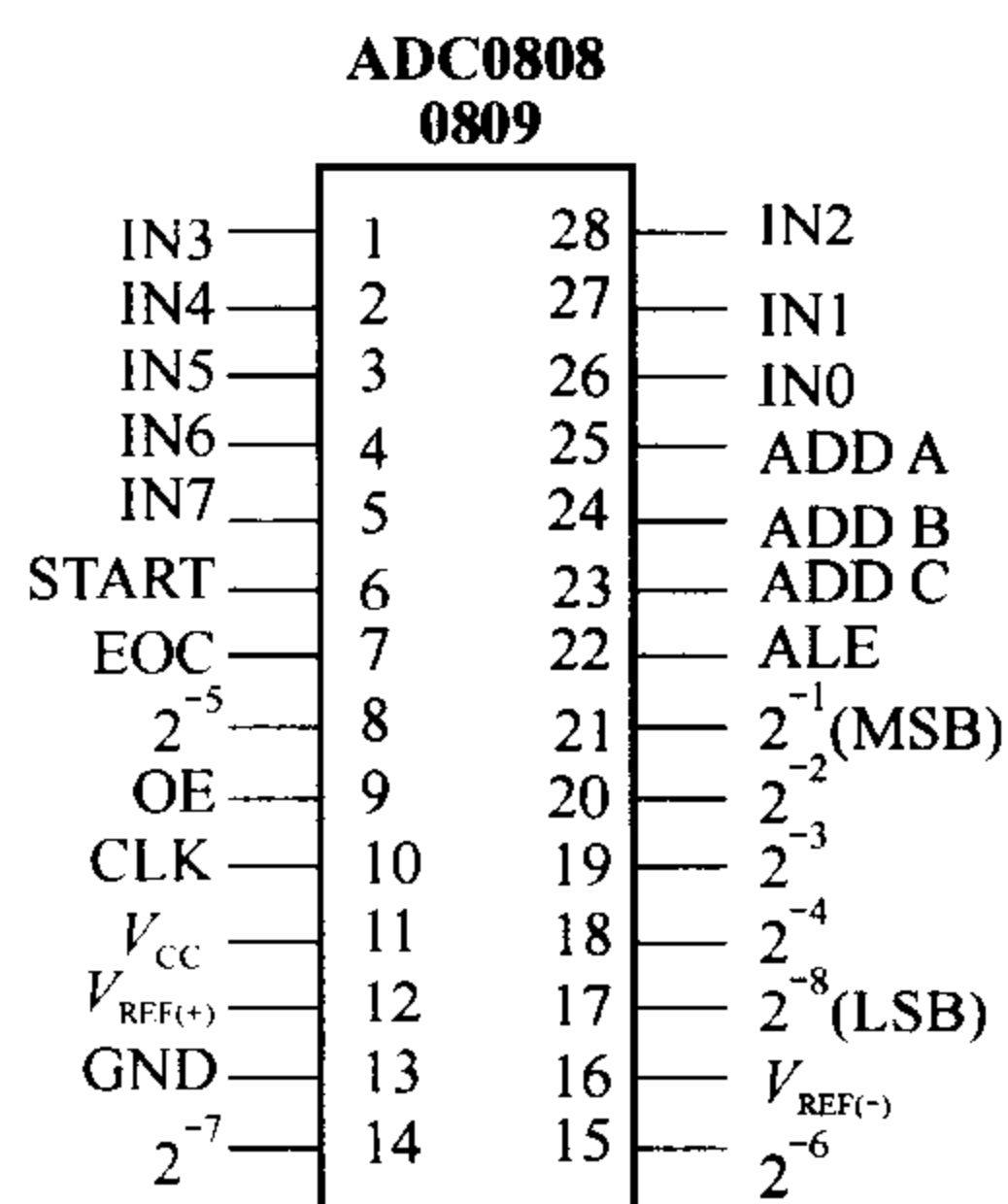


图 10.5.2 ADC0809 引脚图

- (1)  $2^{-1} \sim 2^{-8}$ : 8 位数字量输出引脚。 $2^{-1}$  为最高有效位,  $2^{-8}$  为最低有效位。
- (2) IN0~IN7: 8 路模拟量输入引脚。
- (3)  $V_{REF(+)}$ : 参考电压正端。
- (4)  $V_{REF(-)}$ : 参考电压负端。
- (5) START: A/D 转换启动信号输入端。
- (6) ALE: 地址锁存允许信号接入端。START 和 ALE 两信号用于启动 A/D 转换。
- (7) EOC: 转换结束信号输出引脚。开始转换时为低电平, 转换结束时为高电平。
- (8) OE: 输出允许控制端, 用以打开三态数据输出锁存器。
- (9) CLK: 时钟信号输入端。
- (10) A, B, C: 地址输入线。经译码后可选通 IN0~IN7 这 8 个通道中的一个通道进行转换。A, B, C 的输入与被选通的通道的关系如表 10.5.1 所示。

#### 3. ADC0809 与 AT89S52 接口

##### (1) 查询方式

由于 ADC0809 片内无时钟, 可利用 AT89S52 提供的地址锁存允许信号 ALE 经 D

触发器二分频获得。ALE 引脚的频率是 AT89S52 时钟频率的  $1/6$ , 但要注意的是, 每当访问外部数据存储器时, 将丢失一个 ALE 脉冲。如果单片机的时钟频率采用 6 MHz, 则 ALE 脚的输出频率为 1 MHz, 再分频后为 500 kHz, 恰好符合 ADC0809 对时钟频率的要求。

表 10.5.1 A,B,C 的输入与被选通的通道的关系

被选通的通道	C B A	被选通的通道	C B A
IN0	0 0 0	IN4	1 0 0
IN1	0 0 1	IN5	1 0 1
IN2	0 1 0	IN6	1 1 0
IN3	0 1 1	IN7	1 1 1

图 10.5.3 为 ADC0809 与 AT89S52 单片机的接口电路。由于 ADC0809 具有输出三态锁存器, 数据输出引脚 D7~D0 ( $2^{-1} \sim 2^{-8}$ ) 可直接与数据总线相连, 地址译码引脚 A, B, C 分别与地址总线 A0, A1, A2 (即 P0.0~P0.2) 相连, 以选通 IN0~IN7 中的一个通道。将 P2.7 作为片选信号, 在启动 A/D 转换时, 由单片机的写信号  $\overline{WR}$  和 P2.7 控制 ADC 的地址锁存和转换启动。由于 ALE 和 START 连在一起, 因此 ADC0809 在锁存通道地址的同时, 启动并进行转换。在读取转换结果时, 用低电平的读信号  $\overline{RD}$  和 P2.7 脚经或非门后, 产生的正脉冲作为 OE 信号, 用以打开三态输出锁存器。由图 10.5.3 可知, ADC0809 的 ALE, START, OE 信号的逻辑关系为

$$\text{ALE} = \text{START} = \overline{\overline{\text{WR}} + \text{P2.7}}$$

$$\text{OE} = \overline{\overline{\text{RD}} + \text{P2.7}}$$

可见 P2.7 应设置为低电平。

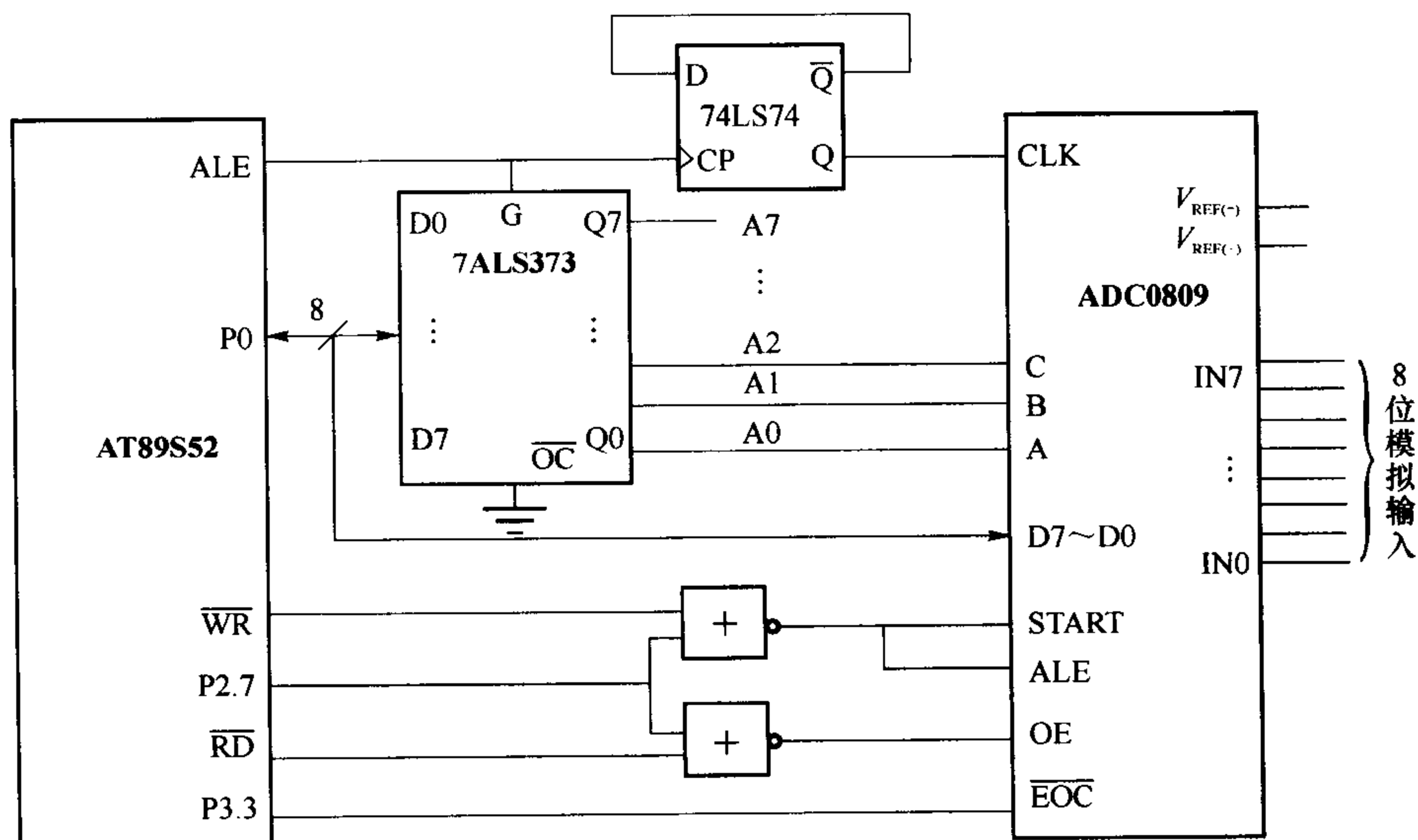


图 10.5.3 ADC0809 与 AT89S52 的接口

由上述分析可知,在编写软件时,应令  $P2.7=0$ ,  $A0, A1, A2$  给出被选择模拟通道的地址。执行一条写出指令,可启动 A/D 转换;执行一条读入指令,可读取转换结果。

转换结束信号 EOC 连接到 AT89S52 的 P3.3 引脚,通过查询 P3.3 的状态判断 A/D 转换是否结束。

下面的程序是采用查询方法,分别对 8 路模拟信号轮流采样一次,并依次把结果存储到内部 RAM 中的采样转换程序:

```

MAIN:  MOV  R1, #DATA      ;数据区地址指针指向首单元
        MOV  DPTR, #7FF8H  ;送通道 0 地址
        MOV  R7, #08H      ;通道数计数器
LOOP:   MOVX  @DPTR, A      ;启动 A/D 转换,  $P2.7=0$  且  $\overline{WR}=0$ 
LOOP1:  NOP
        JNB  P3.3, LOOP1   ;查询转换结束信号 EOC
        MOVX  A, @DPTR     ;读取转换结果,  $P2.7=0$  且  $\overline{RD}=0$ 
        MOV  @R1, A        ;存转换结果
        INC  DPTR          ;指向下一通道
        INC  R1             ;修改数据区指针
        DJNZ R7, LOOP      ;8 个通道全采样完否?
        :

```

## (2) 中断方式

ADC0809 与 AT89S52 的中断方式接口电路只需要将图 10.5.3 中 ADC0809 的 EOC 脚经过一个非门再接到 AT89S52 的  $\overline{INT1}$  脚即可。采用中断方式可大大节省 CPU 的时间。当转换结束时,EOC 发出一个正脉冲,经非门后向单片机申请中断,单片机响应中断请求,由外部中断 1 的中断服务程序读 A/D 转换结果,并启动 ADC0809 的下一次转换。外中断 1 采用下降沿触发方式。

初始化程序如下:

```

INT1:   SETB  IT1          ;外中断 1 初始化
        SETB  EA
        SETB  EX1
        MOV   DPTR, #7FF8H ;启动 ADC0809 对通道 IN0 转换
        MOV   A, #00H      ;A 中可以是任意数值
        MOVX  @DPTR, A
        :

```

中断服务程序清单如下:

```

PINT1:  MOV   DPTR, #7FF8H ;读取 A/D 转换结果,送缓冲单元 30H
        MOVX  A, @DPTR
        MOV   30H, A
        MOV   A, #00H      ;启动 A/D 转换
        MOVX  @DPTR, A
        RETI

```



### 10.5.2 12 位并行 A/D 转换器 AD574 的扩展

AD574 是 AD 公司生产的 12 位逐次逼近型 ADC, 转换速度为  $25\ \mu\text{s}$ , 转换精度为  $0.05\%$ , 由于芯片内有三态输出缓冲电路, 因而可直接与单片机的数据总线相连, 而无须附加逻辑接口电路, 且能与 CMOS 及 TTL 电路兼容。

#### 1. AD574 的特点

AD574 内部结构和 ADC0809 相似, 但位数提高到了 12 位。其特点如下。

- 内部集成有转换时钟, 参考电压源和三态输出锁存器, 可以和单片机直接接口, 无须外接 CLOCK 时钟。
- 输入模拟电压既可以是单极性也可以是双极性; 单极性输入时为  $0\sim+10\text{ V}$  或  $0\sim+20\text{ V}$ ; 双极性输入时可在  $\pm 5\sim\pm 10\text{ V}$  之间。
- 数字量位数可设定为 8 位, 也可设定为 12 位。

#### 2. AD574 的引脚及功能

28 脚双列直插式封装的引脚如图 10.5.4 所示。其引脚功能说明如下。

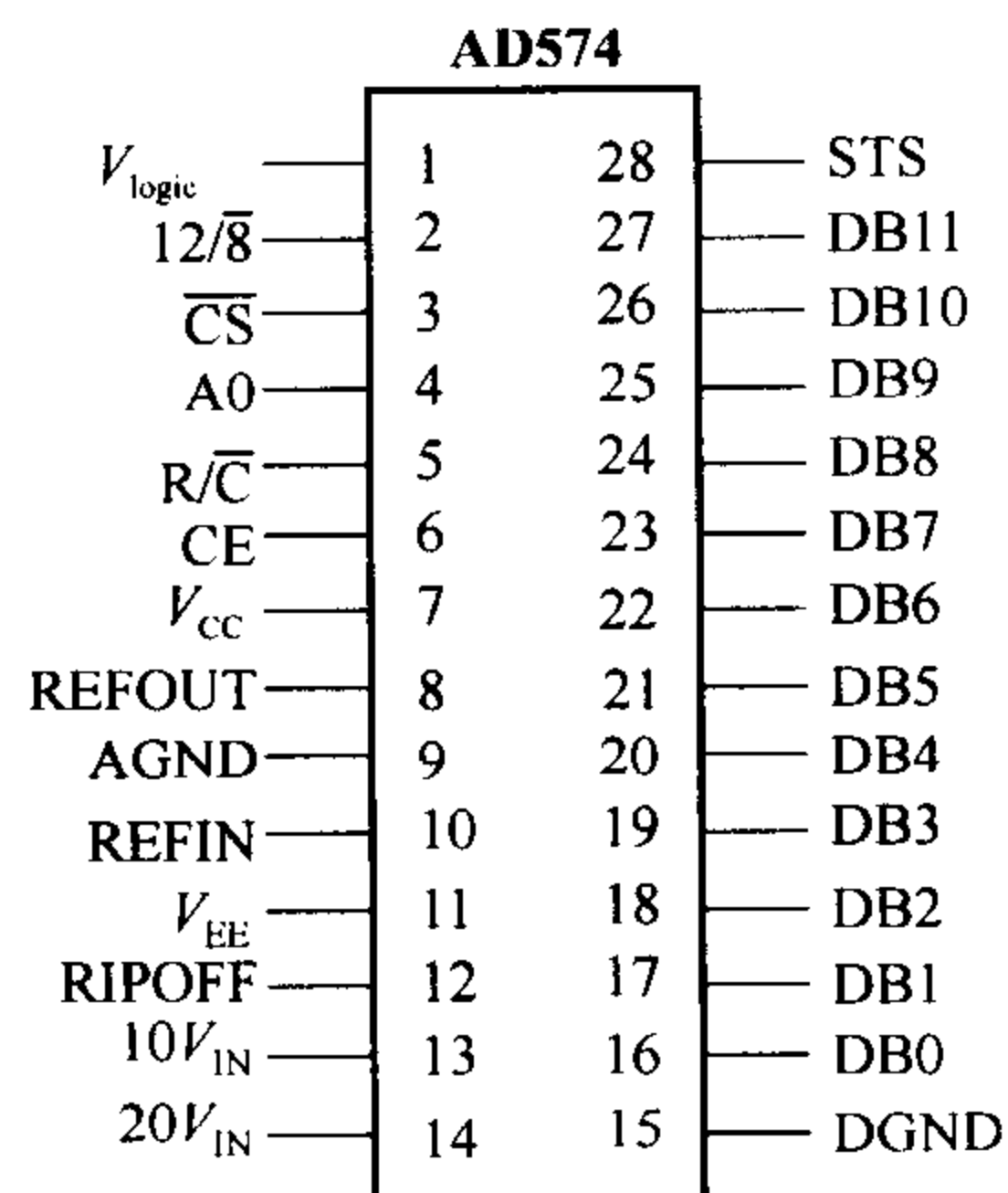


图 10.5.4 AD574 引脚图

##### (1) 模拟量输入引脚

$10V_{\text{IN}}$ :  $10\text{ V}$  量程模拟电压输入引脚, 单极性时为  $0\sim+10\text{ V}$ , 双极性时为  $-5\sim+5\text{ V}$ 。

$20V_{\text{IN}}$ :  $20\text{ V}$  量程模拟电压输入引脚, 单极性时为  $0\sim+20\text{ V}$ , 双极性时为  $-10\sim+10\text{ V}$ 。

AGND: 模拟电压公共地引脚。

##### (2) 数字量输出引脚

DB0~DB11: 数字量输出引脚。

DGND: 数字量公共接地引脚。

##### (3) 控制引脚

$\overline{\text{CS}}$ : 片选信号, 低电平有效。

CE: 片允许信号, 高电平有效。简单应用可接高电平 ( $+5\text{ V}$ )。

$\text{R}/\overline{\text{C}}$ : 读出/转换控制信号,  $\text{R}/\overline{\text{C}}=0$  启动 A/D 转换;  $\text{R}/\overline{\text{C}}=1$  允许读出数字量。

$12/\overline{8}$ : 数据输出格式选择信号引脚。当  $12/\overline{8}=1(+5\text{ V})$  时, 12 条数据线同时并行输出; 当  $12/\overline{8}=0(0\text{ V})$  时, 8 位双字节输出。注意该脚不能接 TTL 信号, 只能接至  $+5\text{ V}$  或 GND。

A0: 字节选择线。在转换期间, 当  $A0=0$  时, AD574 进行全 12 位转换, 转换时间为  $25\ \mu\text{s}$ ; 当  $A0=1$  时, 进行 8 位转换, 转换时间为  $16\ \mu\text{s}$ 。在读出期间, 当  $A0=0$  时, 输出高 8 位; 当  $A0=1$  时, 输出低 4 位, 并以 4 个 0 作为尾随的 4 位补足 8 位, 即当两次读出 12 位数据时, 应遵循左对齐原则。

STS: 转换状态信号引脚。转换开始时, STS 达到高电平, 转换过程中保持高电平。转换完成时变为低电平。STS 可以作为状态信息被 CPU 查询, 也可以用它的下降沿向 CPU 发出中断请求, 通知 A/D 转换已完成, CPU 可以读出转换结果。

AD574 的控制信号真值表如表 10.5.2 所示。

表 10.5.2 AD574 控制信号真值表

CE	$\overline{\text{CS}}$	R/ $\overline{\text{C}}$	12/8	A0	操 作
0	×	×	×	×	禁止
×	1	×	×	×	禁止
1	0	0	×	0	启动 12 位转换
1	0	0	×	1	启动 8 位转换
1	0	1	5 V	×	12 位并行输出格式
1	0	1	0 V	0	高 8 位输出格式
1	0	1	0 V	1	低 4 位+4 位尾 0 输出格式

通过改变 AD574 引脚 8、10、12 的外接电路,可使 AD574 进行单极性和双极性模拟输入电压的 A/D 转换。可实现输入信号 0~10 V 或 0~20 V 的转换,也可实现输入信号 -5~+5 V 或 -10~+10 V 的转换。

### 3. AT89S52 与 AD574 的接口

图 10.5.5 为 AD574 与 AT89S52 单片机的接口电路。由于 AD574 片内有时钟,故无须外加时钟信号。该电路采用单极性输入方式,可对 0~10 V 或 0~20 V 模拟信号进行转换。转换结果的高 8 位从 D11~D4 输出,低 4 位从 D3~D0 输出,并且直接和单片机的数据总线相连。遵循左对齐原则,D3~D0 应接单片机数据总线的高半字节(P0.7~P0.4)。为了启动 A/D 转换和读出转换结果,AD574 的片选信号  $\overline{\text{CS}}$  由地址总线的次低位 A1(P0.1)提供,读写时 A1 应设置为低电平。AD574 的 CE 信号由单片机的  $\overline{\text{WR}}$  和 A7(P0.7)经一级或非门产生。R/ $\overline{\text{C}}$  则由  $\overline{\text{RD}}$  和 A7 经一级或非门提供。在读写时,A7 亦应为低电平。输出状态信号 STS 接 P3.2 端供单片机查询,以判断 A/D 转换是否结束。12/8 端接地,AD574 的 A0 由地址总线的最低位 A0(P0.0)控制,以实现全 12 位转换,将 12 位数据分两次送入数据总线上。

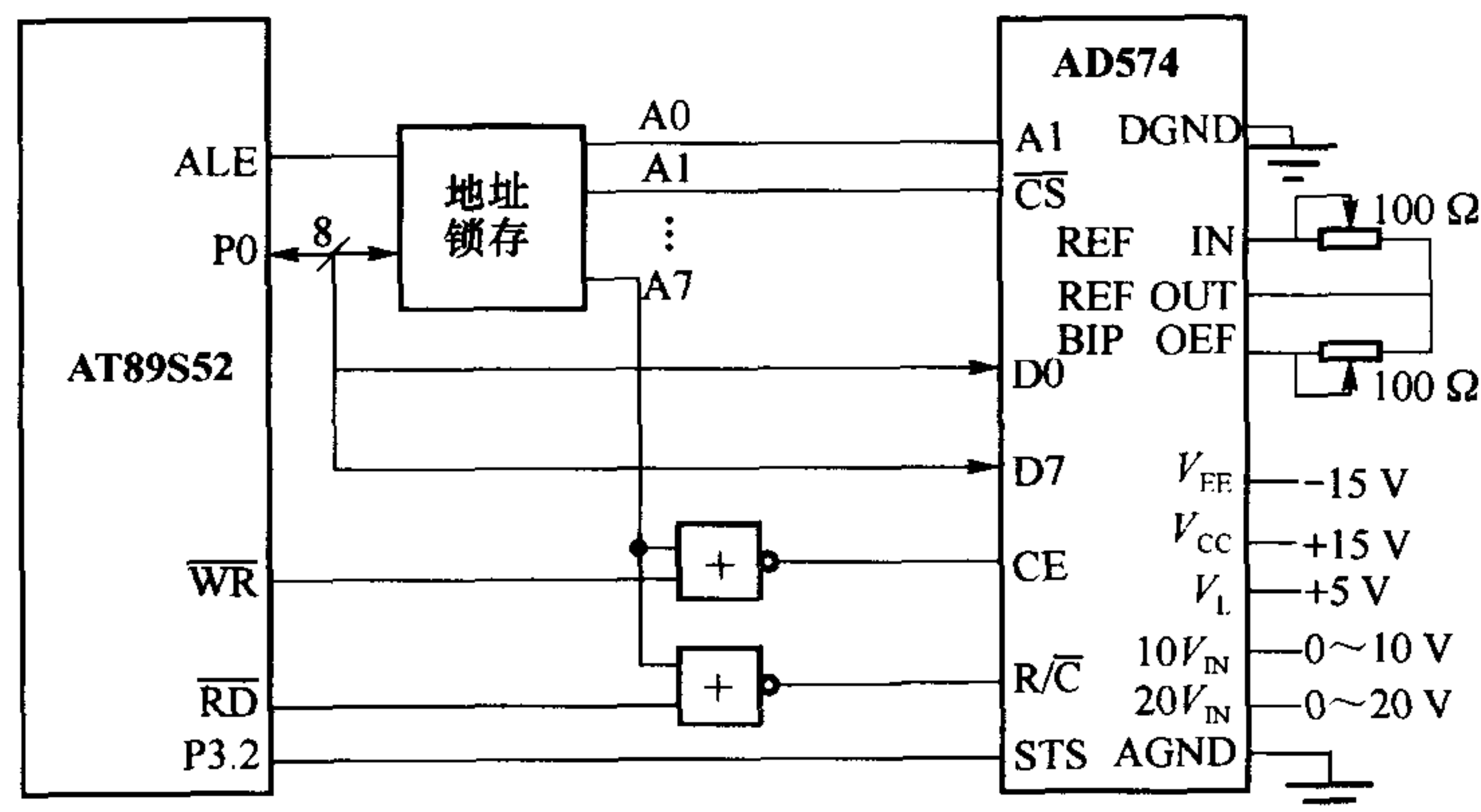


图 10.5.5 AD574 与 AT89S52 的接口

利用该接口电路完成一次 A/D 转换,并把转换结果高 8 位存入 R2 中,低 8 位存入 R3 中的工作程序如下:

```

MIAN:  MOV    R0, #7CH    ;选择 AD574,并令 A0 = 0
        MOVX   @R0,A      ;启动 A/D 转换,全 12 位
LOOP:   NOP
        JB     P3.2,LOOP   ;查询转换是否结束
        MOVX   A,@R0      ;读取高 8 位
        MOV    R2,A       ;存入 R2 中
        MOV    R0, #7DH   ;令 A0 = 1
        MOVX   A,@R0      ;读取低 4 位,尾随 4 个 0
        MOV    R3,A       ;存入 R3 中
        :

```

### 10.5.3 12 位串行 A/D 转换器 TLC2543 的扩展

TLC2543 是德州仪器公司开发的一种 11 通道输入,数据输出长度可调的 12 位串行 A/D,它以很少的引脚提供快速、高精度的 A/D 转换,同时还具有采样保持电路。

#### 1. TLC2543 的特点

- 电源电压:5 V。
- 11 通道输入,12 位 A/D 转换器,在工作温度范围内转换时间为 10  $\mu$ s。
- 3 种内建的自检模式,内置采样保持电路。
- 最大 1/4 096 的线性误差。
- 内置系统时钟,转换结束标志。
- 单/双极性输出,可支持软件关机。
- 输入输出的顺序可编程(高位还是低位在前),输出数据长度可编程。

#### 2. TLC2543 的引脚及逻辑结构

TLC2543 为 20 引脚,有双列直插和方形贴片两种,引脚和内部结构如图 10.5.6 所示。

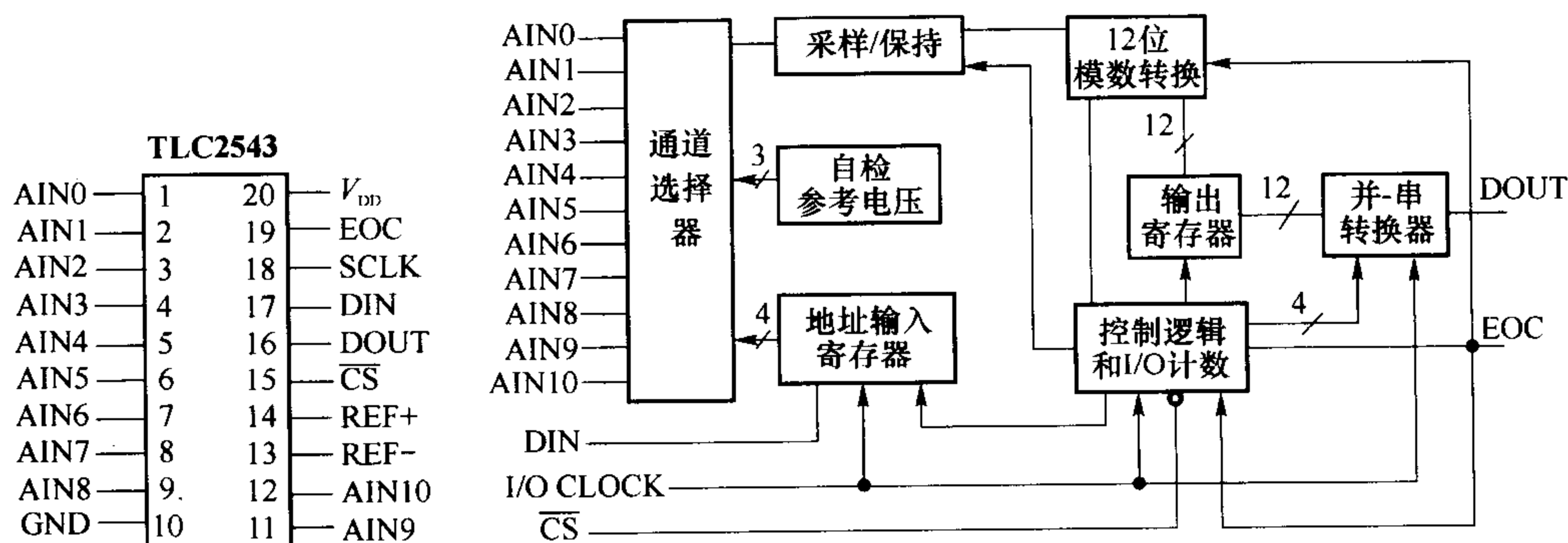


图 10.5.6 TLC2543 引脚和内部结构

引脚说明如表 10.5.3 所示:

表 10.5.3 TLC2543 引脚说明

引脚名称	说 明	引脚名称	说 明
AIN0~AIN10	模拟输入通道	EOC	转换结束信号
$\overline{\text{CS}}$	片选	SCLK(I/O CLOCK)	输入输出同步时钟
DIN	串行数据输入	RE5F+	正参考电压
DOUT	转换结束数据输出	REF—	负参考电压

TLC2543 内部由通道选择器、采样保持电路、12 位 A/D 转换器、输出寄存器、并行到串行转换器以及控制逻辑电路组成。通道选择器根据输入地址寄存器中存放的模拟输入通道地址,选择输入通道,并将输入通道中的信号送到采样保持电路中,然后在 12 位 A/D 转换器中将采样的模拟量进行量化编码,转换成数字量,存放到输出寄存器中,这些数据经过并行到串行转换器转换成串行数据,经 TLC2543 的 DOUT 端输出到单片机中。

### 3. TLC2543 的命令字

TLC2543 每次转换都要写入命令字,以便确定下一次转换用哪一通道,下次转换结果用多少位输出,转换结果输出是用低位在前还是高位在前。命令字的输入高位在前,其格式如表 10.5.4 所示。

表 10.5.4 TLC2543 的命令字

通道选择位	输出数据长度控制位	输出顺序控制位	极性选择位
D7 D6 D5 D4	D3 D2	D1	D0

通道选择位 D7,D6,D5,D4:用以选择模拟输入通道。

输出数据长度控制位 D3,D2:控制下次的转换结果用多少位输出。D3D2=01 时,8 位输出;D3D2=×0 时,12 位输出,×表示“不关心”条件,可以为 0 或 1;D3D2=11 时,16 位输出。由于转换器的分辨率为 12 位,因而建议用 12 位长度。

输出顺序控制位 D1:控制转换结果的输出顺序。D1=0 时,高位在前;D1=1 时,低位在前。

极性选择位 D0:输出数据符号位选择。D0=0 时,输出数据为单极性(无符号二进制)数据;D0=1 时,输出数据为双极性(有符号二进制)数据。

### 4. TLC2543 的 SPI 接口时序

TLC2543 的 SPI 接口时序如图 10.5.7 所示。

### 5. AT89S52 与 TLC2543 的 SPI 接口

(1) SPI(Serial Peripheral Interface)是一种串行接口标准,串行通信的双方用 4 条线进行通信,分别是:片选信号线、串行输入、串行输出和 I/O 时钟。

## (2) TLC2543 与 AT89S52 接口及程序

图 10.5.8 是 AT89S52 与 TLC2543 的连接电路。SPI 方式是 TLC2543 和微处理器之间进行数据传送的最快和最有效的方法。AT89S52 单片机不带 SPI 接口,为了和 TLC2543 接口,需要用软件来合成 SPI 的操作。由于受指令周期的影响,数据的传送速率下降。因此为提高接口的数据传送速率,要尽可能提高系统的时钟频率。

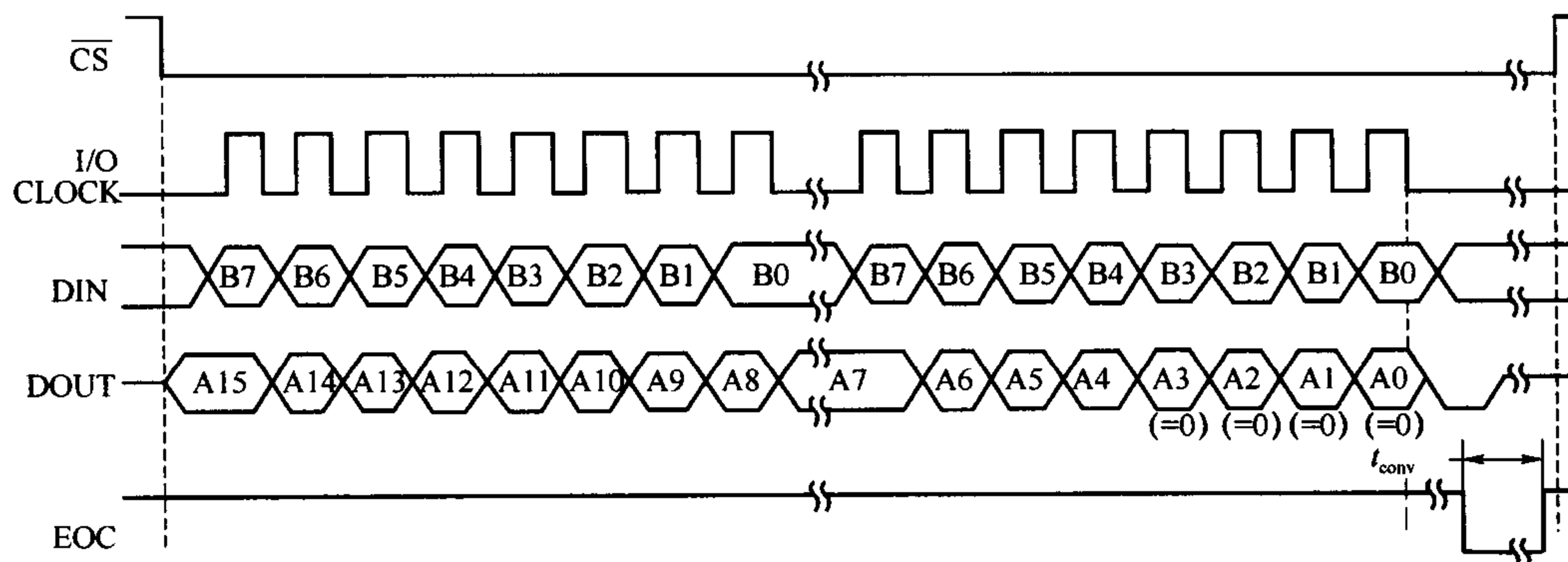


图 10.5.7 TLC2543 的 SPI 接口时序图

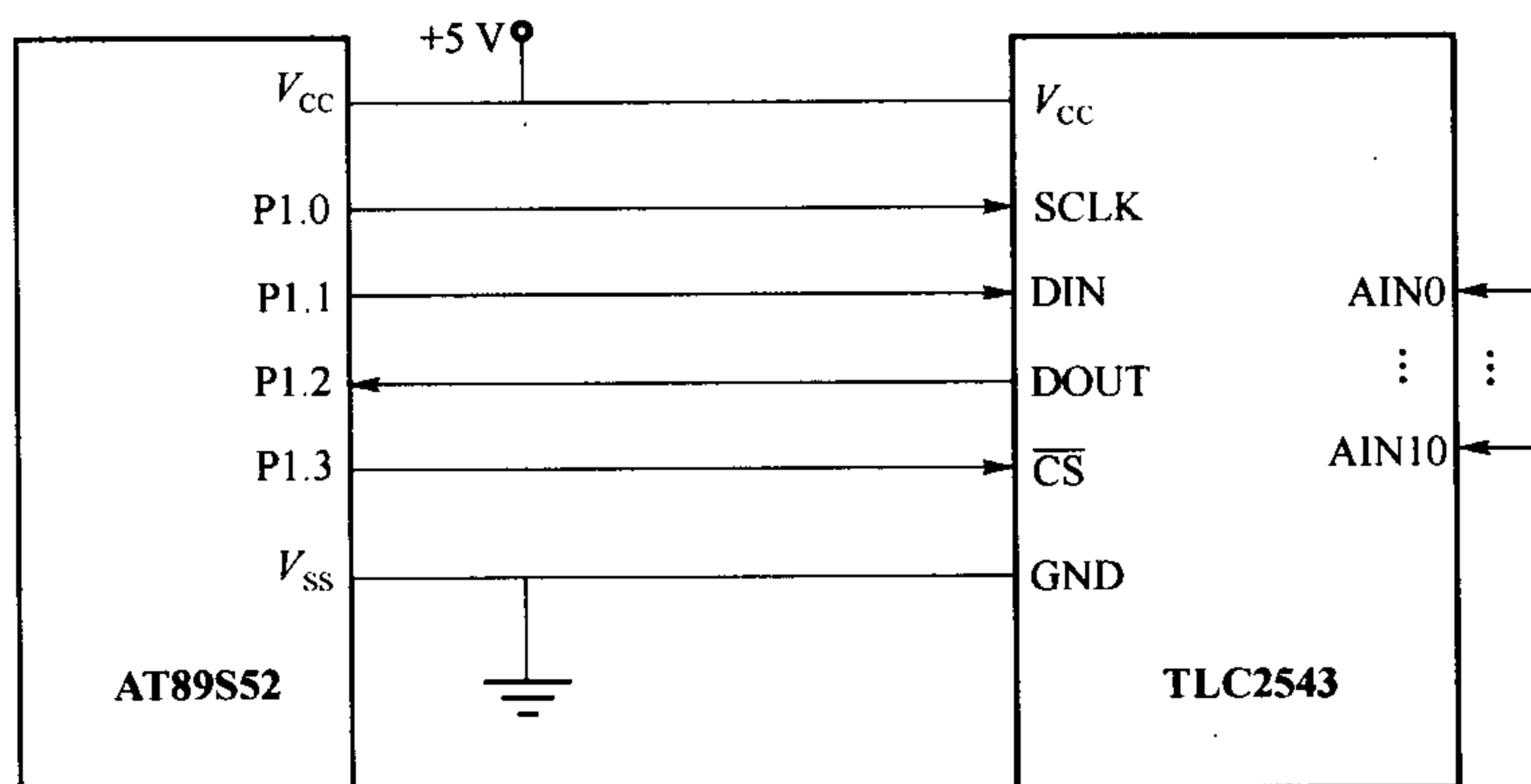


图 10.5.8 AT89S52 和 TLC2543 连接图

AT89S52 的 P1.3 口用于控制  $\overline{CS}$ , P1.2 作为数据输入线, P1.1 作为命令输出线, P1.0 提供 I/O CLK, 通过对 P1.0 编程产生时钟。AT89S52 将用户的命令字通过 P1.1 输入到 TLC2543 中, 等待  $20\mu s$  开始读数据, 同时写下一次的命令字。

**例 10-7** 利用 TLC2543 的 AIN0 采集 10 个数据, 放入到 AT89S52 内部 RAM 的 30H 开始的单元中。单片机采用 12 MHz 的晶振, 数据格式为 12 位、高位在前、单极性。

**解** 命令字为 00H。程序清单如下。

```
MOV    P1, #04H      ;准备读 P1.2
MOV    R6, #0AH      ;转换 10 次
MOV    R0, #2FH      ;置数据缓冲区指针
```

```

CLR    P1.0           ;置 I/O 时钟为低
SETB   P1.3           ;置  $\overline{CS}$  为高
ACALL  TLC2543        ;调转换子程序
SJMP   $

```

转换子程序:

```

TLC2543:MOV    A,#0H      ;通道选择和工作模式送 A
          CLR    P1.3      ;置  $\overline{CS}$  为低
          MOV    R5,#0CH   ;置输出位计数初值
LOOP:     MOV    C,P1.2    ;读入转换数据一位
          RLC     A         ;将进位位右移给 A,(将转换数据的一位读入,
                           ;同时将一位控制位移入 C)
          MOV    P1.1,C    ;送出一位控制位
          SETB   P1.0      ;置 I/O 时钟为高
          CLR    P1.0      ;置 I/O 时钟为低
          CJNE   R5,#05H   ;LOP1
          MOV    @R0,A     ;前 8 位存入 RAM
          INC    R0
          CLR    A
LOP1:     DJNZ   R5,LOOP    ;未转换完继续
          ANL    A,#0FH
          MOV    @R0,A     ;转换完的存入单元
          INC    R0
          MOV    R2,#0AH   ;延时
DELAY:    DJNZ   R2,DELAY
          DJNZ   R6,TLC2543
          RET

```

#### 10.5.4 8 位并行 D/A 转换器 DAC0832 的扩展

##### 1. DAC0832 的结构原理

###### (1) DAC0832 的特性

美国国家半导体公司的 DAC0832 芯片是具有两级输入数据寄存器的 8 位单片 D/A 转换器,能直接与 AT89S52 单片机相连接,其主要特性如下。

DAC0832 采用二次缓冲方式,可以在输出的同时,采集下一个数据,从而提高转换速度;能够在多个转换器同时工作时,实现多通道 D/A 的同步转换输出。

8 位分辨率,电流输出,稳定时间为  $1\ \mu\text{s}$ ;可双缓冲、单缓冲或直接数字输入;只需在满量程下调整其线性度;单一电源供电( $+5\sim+15\ \text{V}$ );低功耗,20 mW;逻辑电平输入与 TTL 兼容。

## (2) DAC0832 的引脚及逻辑结构

DAC0832 的引脚如图 10.5.9 所示。其逻辑结构如图 10.5.10 所示,由 8 位锁存器、8 位 DAC 寄存器和 8 位 D/A 转换器构成。

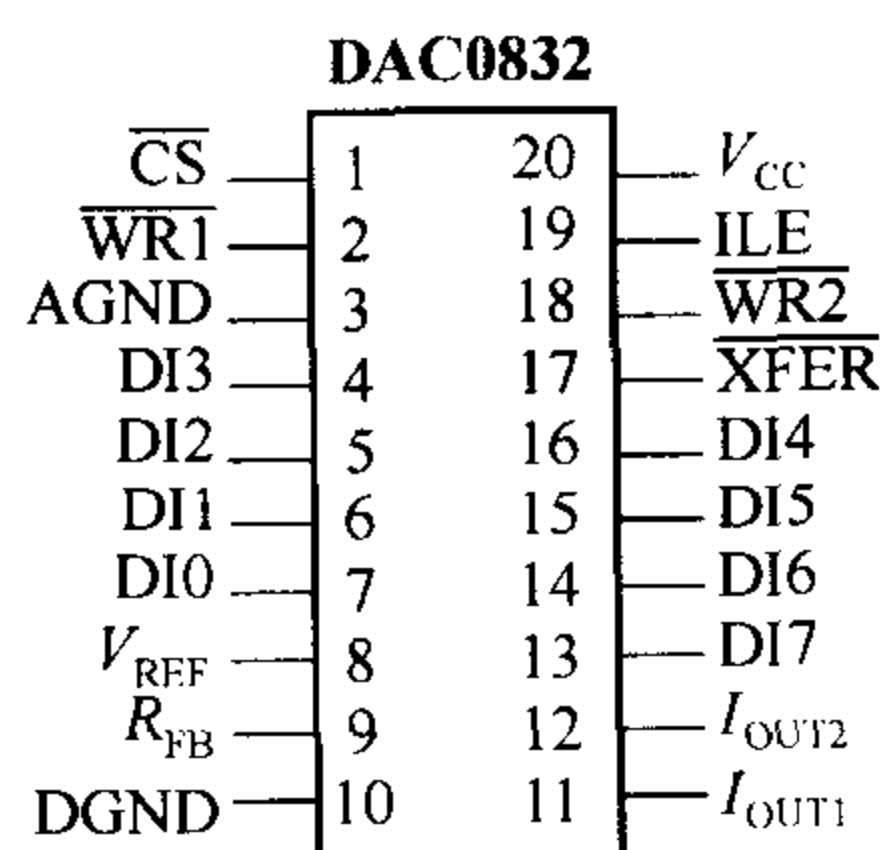


图 10.5.9 DAC0832 芯片的引脚

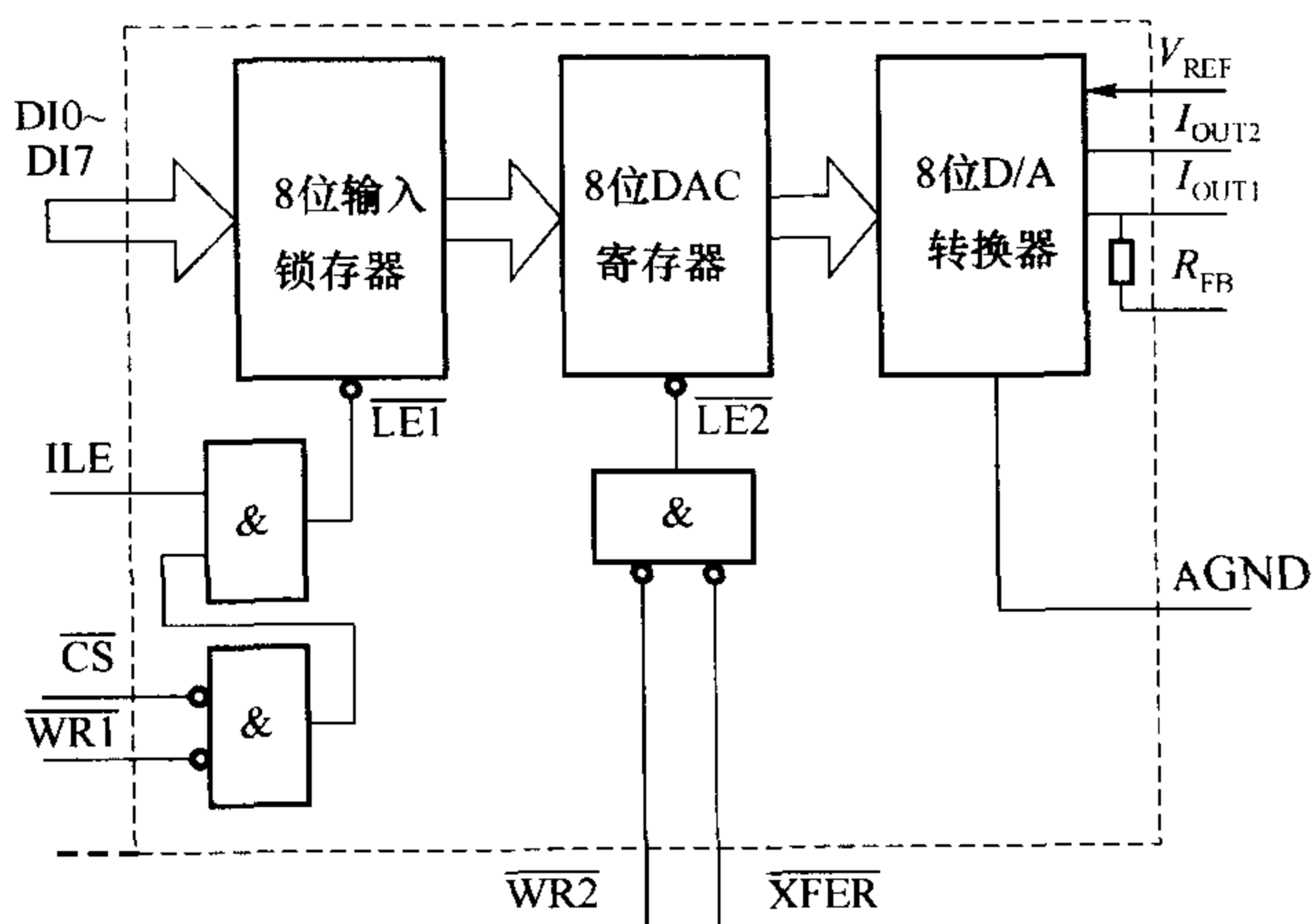


图 10.5.10 DAC0832 的逻辑结构框图

DAC0832 各引脚的功能说明如下。

DI0 ~ DI7: 转换数据输入线。

ILE: 数据允许锁存信号, 高电平有效。

$\overline{\text{CS}}$ : 输入寄存器选择信号, 低电平有效。

$\overline{\text{WR1}}$ : 输入寄存器写选通信号。

$\overline{\text{LE1}}$ : 输入寄存器的锁存信号。 $\overline{\text{LE1}}$  为高电平时, 输入锁存器的状态随输入数据线的状态变化;  $\overline{\text{LE1}}$  的下降沿将输入数据线上的数据存入输入锁存器。 $\overline{\text{LE1}}$  由 ILE、 $\overline{\text{CS}}$ 、 $\overline{\text{WR1}}$  的逻辑组合产生。

$$\overline{\text{LE1}} = (\overline{\text{WR1}}) + (\overline{\text{CS}}) \cdot \text{ILE}$$

当 ILE 为高电平,  $\overline{\text{CS}}$  为低电平,  $\overline{\text{WR1}}$  输入负脉冲时,  $\overline{\text{LE1}}$  产生正脉冲。

$\overline{\text{XFER}}$ : 数据传送信号, 低电平有效。

$\overline{\text{WR2}}$ : DAC 寄存器的写选通信号。

$\overline{\text{LE2}}$ : DAC 寄存器的锁存信号。 $\overline{\text{LE2}}$  为高电平时, DAC 寄存器的输出和输入锁存器的状态一致,  $\overline{\text{LE2}}$  的下降沿将输入寄存器的内容存入 DAC 寄存器。 $\overline{\text{LE2}}$  由  $\overline{\text{XFER}}$ 、 $\overline{\text{WR2}}$  的逻辑组合产生。

$$\overline{\text{LE2}} = (\overline{\text{WR2}}) + (\overline{\text{XFER}})$$

当  $\overline{\text{XFER}}$  为低电平,  $\overline{\text{WR2}}$  输入负脉冲时, 则在  $\overline{\text{LE2}}$  产生正脉冲。

$V_{\text{REF}}$ : 基准电源输入引脚。

$R_{\text{FB}}$ : 反馈信号输入引脚, 反馈电阻在芯片内部。

$I_{\text{OUT1}}$ ,  $I_{\text{OUT2}}$ : 电流输出引脚。电流  $I_{\text{OUT1}}$  和  $I_{\text{OUT2}}$  的和为常数,  $I_{\text{OUT1}}$ ,  $I_{\text{OUT2}}$  随 DAC 寄存器的内容线性变化。 $I_{\text{OUT1}} = \frac{V_{\text{REF}}}{15 \text{ k}\Omega} \times \frac{D}{256}$ ,  $I_{\text{OUT2}} = \frac{V_{\text{REF}}}{15 \text{ k}\Omega} \times \frac{255-D}{256}$ , 其中, 15 k $\Omega$  是芯片内

电阻网络的标称值,  $D$  为要转换的数字量。

$V_{CC}$ : 电源输入引脚。

AGND: 模拟信号地。

DGND: 数字信号地。

DAC0832 是电流输出型。在单片机应用系统中, 通常需要电压信号, 电流信号到电压信号的转换由运算放大器实现。

## 2. AT89S52 与 D/A 转换器接口

从图 10.5.10 DAC0832 的逻辑结构框图中可以总结出 DAC0832 控制信号的逻辑关系:

- 当  $\overline{CS}=0$ ,  $\text{ILE}=1$  时,  $\overline{WR1}$  信号有效时将数据总线上的信号写入 8 位输入锁存器;
- 当  $\overline{XFER}=0$  时,  $\overline{WR2}$  信号有效时将输入寄存器的数据转移到 8 位 DAC 寄存器中, 此时 D/A 转换器的输出随之改变。

根据上述功能, 可以将 DAC0832 连接成直通工作方式、单缓冲工作方式和双缓冲工作方式。

### (1) 直通工作方式应用

当某一条地址线或地址译码器的输出线使 DAC0832 的脚  $\overline{CS}$  有效 (低电平),  $\text{ILE}$  脚高电平,  $\overline{WR1}$ ,  $\overline{XFER}$  和  $\overline{WR2}$  为低电平时, 单片机数据线上的数据字节直通 D/A 转换器, 被转换并输出。

### (2) 单缓冲方式应用

图 10.5.11 为具有一路模拟量输出的应用系统。图 10.5.11 中  $\text{ILE}$  脚接高电平,  $\overline{CS}$  和  $\overline{XFER}$  脚连在一起都接到地址线 P2.7 脚, 输入寄存器和 DAC 寄存器地址都是 7FFFH;  $\overline{WR1}$  和  $\overline{WR2}$  连到一起都和单片机的写信号  $\overline{WR}$  相连。单片机对 DAC0832 执行一次写操作, 则把一个字节数据直接写入 DAC 寄存器中, DAC0832 输出的模拟量随之变化。

$$V_{\text{OUT}} = -(I_{\text{OUT1}} \times R_{\text{FB}}) = -\frac{V_{\text{REF}} \cdot D}{256}$$

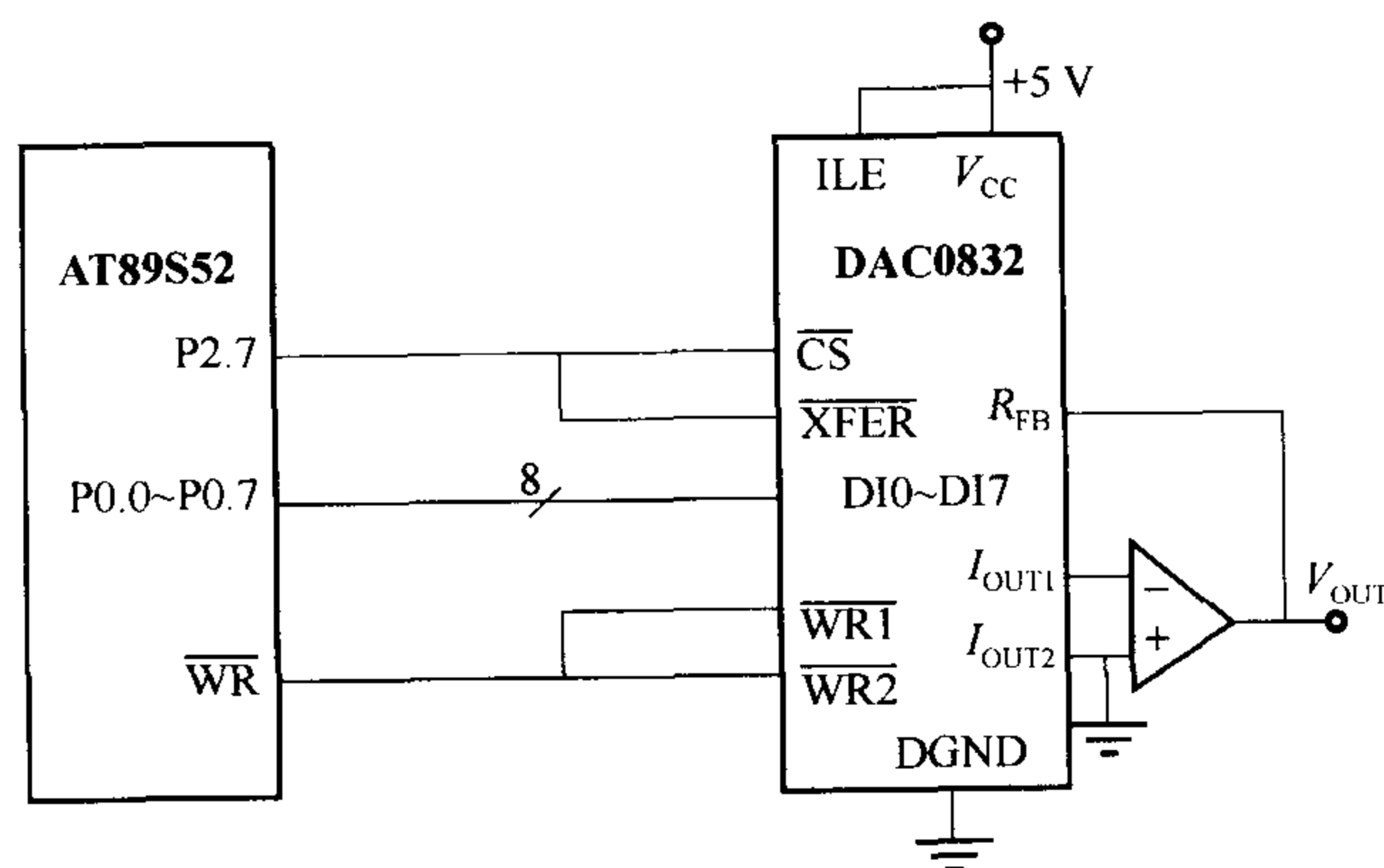


图 10.5.11 DAC0832 单缓冲器方式应用

**例 10-8** 根据图 10.5.10 电路, 编写产生锯齿波和方波程序。

**解** 产生锯齿波程序清单如下:

```
START:    MOV    DPTR, #7FFFH    ; 设置 D/A 口地址
```



	MOV	A, #00H	; 输入数字量初值 00H 到 A
LOOP:	MOVX	@DPTR, A	; 输出对应于 A 内容的模拟量
	INC	A	; 修改 A 的内容
	AJMP	LOOP	; 返回循环

产生方波程序清单:

START:	MOV	DPTR, #7FFFH	; 设置 D/A 口地址
LOOP:	MOV	A, #FFH	; 给 A 送最大值
	MOVX	@DPTR, A	; D/A 输出相应模拟量
	ACALL	delay	; 延时(略)
	MOV	A, #00H	; 给 A 送最小值
	MOVX	@DPTR, A	; D/A 输出相应模拟量
	ACALL	delay	; 延时(略)
	AJMP	LOOP	; 返回循环

### (3) 双缓冲方式应用

对于多路 D/A 转换接口, 要求同步进行 D/A 转换输出时, 必须采用双缓冲方式。DAC0832 数字量输入锁存和 D/A 转换输出是分两步完成的, 即 CPU 的数据总线分时输出数字量并锁存在各 D/A 转换器的输入寄存器中, 然后 CPU 对所有 D/A 转换器发出控制信号, 使各输入寄存器中的数据存入相应的 DAC 寄存器, 实现同步转换输出。

在图 10.5.12 中每一路模拟量输出需一片 DAC0832。DAC0832(1) 输入锁存器地址为 0DFFFH, DAC0832(2) 输入锁存器的地址为 0BFFFH, DAC0832(1) 和 DAC0832(2) 的第二级寄存器地址同为 7FFFH。

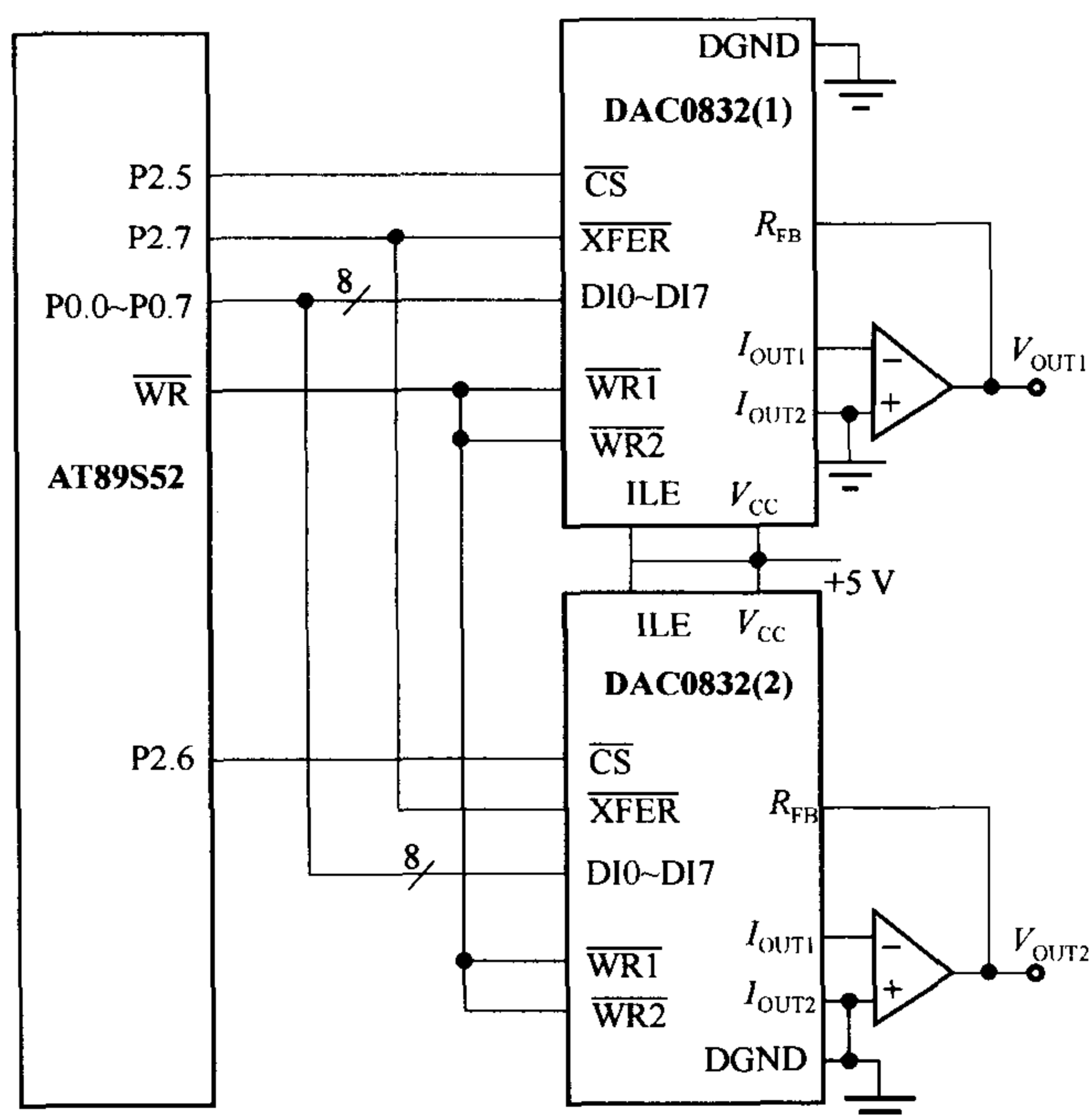


图 10.5.12 DAC0832 双缓冲方式接口电路

**例 10-9** 根据图 10.5.12, 编写程序实现两路 D/A 同步转换输出。

**解** 程序清单如下:

```

MOV    DPTR, #0DFFFH    ;指向 DAC0832(1)
MOV    A, #DATA1        ;DATA1 送入 DAC0832(1) 中锁存
MOVX   @DPTR, A
MOV    DPTR, #0BFFFH    ;指向 DAC0832(2)
MOV    A, #DATA2        ;DATA2 送入 DAC0832(2) 中锁存
MOVX   @DPTR, A
MOV    DPTR, #7FFFH     ;给 0832(1) 和 (2) 提供  $\overline{WR}$  信号
MOVX   @DPTR, A         ;同时完成 D/A 转换输出

```

### 10.5.5 12 位串行 D/A 转换器 TLV5616 的扩展

利用串行 D/A 转换器进行 D/A 转换可以节省单片机的 I/O 口线。本小节介绍 TI 公司的 TLV5616 串行 D/A 转换器。

#### 1. TLV5616 的特性

- 电源电压范围: 2.7~5.5 V。
- 12 位电压输出 DAC。
- 编程控制建立时间: 快速模式为 3  $\mu$ s, 低速模式 9  $\mu$ s。
- 超低功耗; 省电模式(电流为 10 nA)。
- 非线性误差的典型值  $< 0.5$  LSB。
- 与 SPI 串行接口兼容。
- 高阻缓冲的参考电压输入。
- 电压输出为参考电压的 2 倍。

#### 2. TLV5616 的引脚和内部结构(如图 10.5.13 所示)

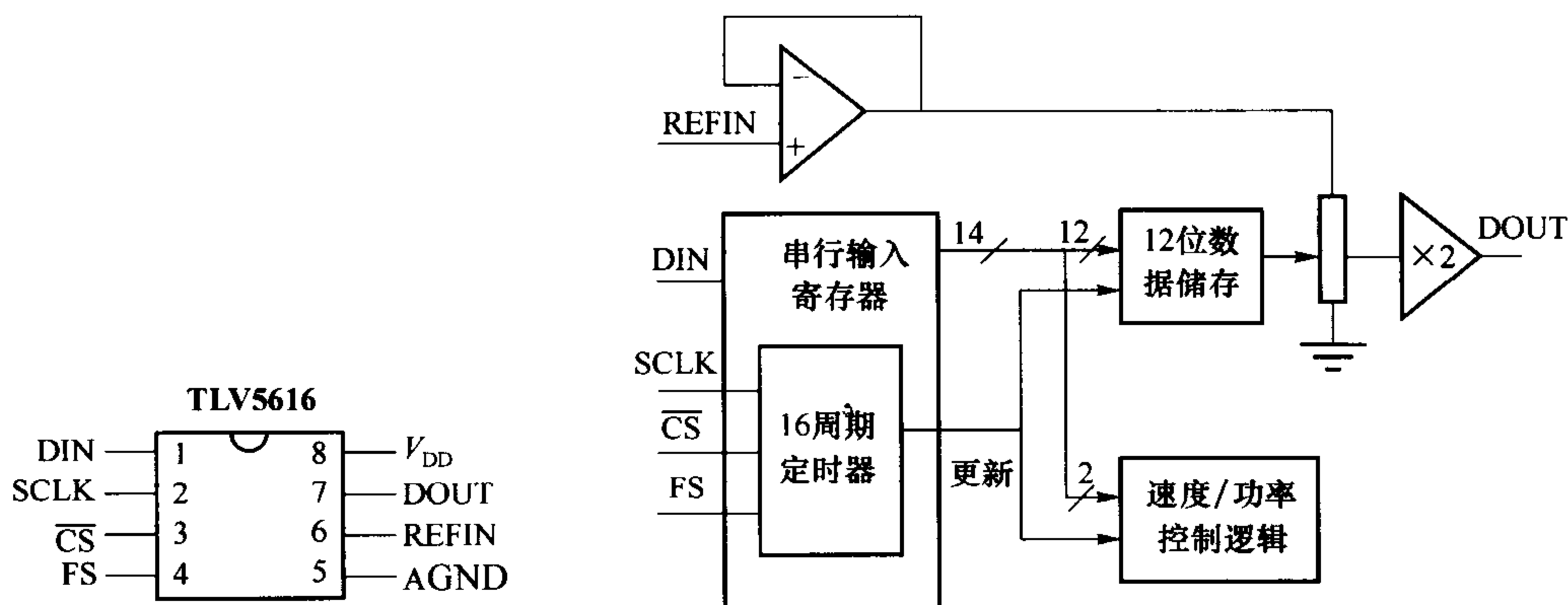


图 10.5.13 TLV5616 的引脚和内部结构

TLV5616 的引脚说明如表 10.5.5 所示。

表 10.5.5 TLV5616 的引脚说明

引脚名称	说 明	引脚名称	说 明
DIN	串行数据输入	DOUT	转换结束数据输出
SCLK	输入同步时钟	FS	帧同步信号
REFIN	输入转换参考电压	$\overline{\text{CS}}$	片选信号

图 10.5.13 中数据由 DIN 在 SCLK 时钟控制下串行移位进入串行输入寄存器。每 16 位为一帧,由 FS 进行帧同步,数据的控制位(D14 和 D13 两位)进入速度和功率下降控制逻辑,对转换速度和功耗进行控制,串行数据中的 12 位待转换数据通过 12 位数据锁存器后进入电阻网络完成 D/A 转换,模拟电压经过 2 倍增益的放大器从 DOUT 输出,参考电压从 REFIN 输入,经高阻抗运放加到电阻网络,这样可以降低功耗。

### 3. TLV5616 的数据格式

D15	D14	D13	D12	D11~D0
×	SPD	PWR	×	待转换的 12 位数字值

其中,×为“不关心”位。

SPD 为速度控制位,1 为快速,0 为慢速。

PWR 为功耗控制位,1 为低功耗,0 为正常功耗。

D11~D0 为即将进行转换的 12 位数字值。

输出模拟电压 =  $2V_{\text{REF}} \times \frac{D}{2^{12}-1}$ , 其中  $V_{\text{REF}}$  为参考电压,  $D$  为转换数字量。

### 4. 时序图

TLV5616 的读写时序图如图 10.5.14 所示。图中参数如表 10.5.6 所示。

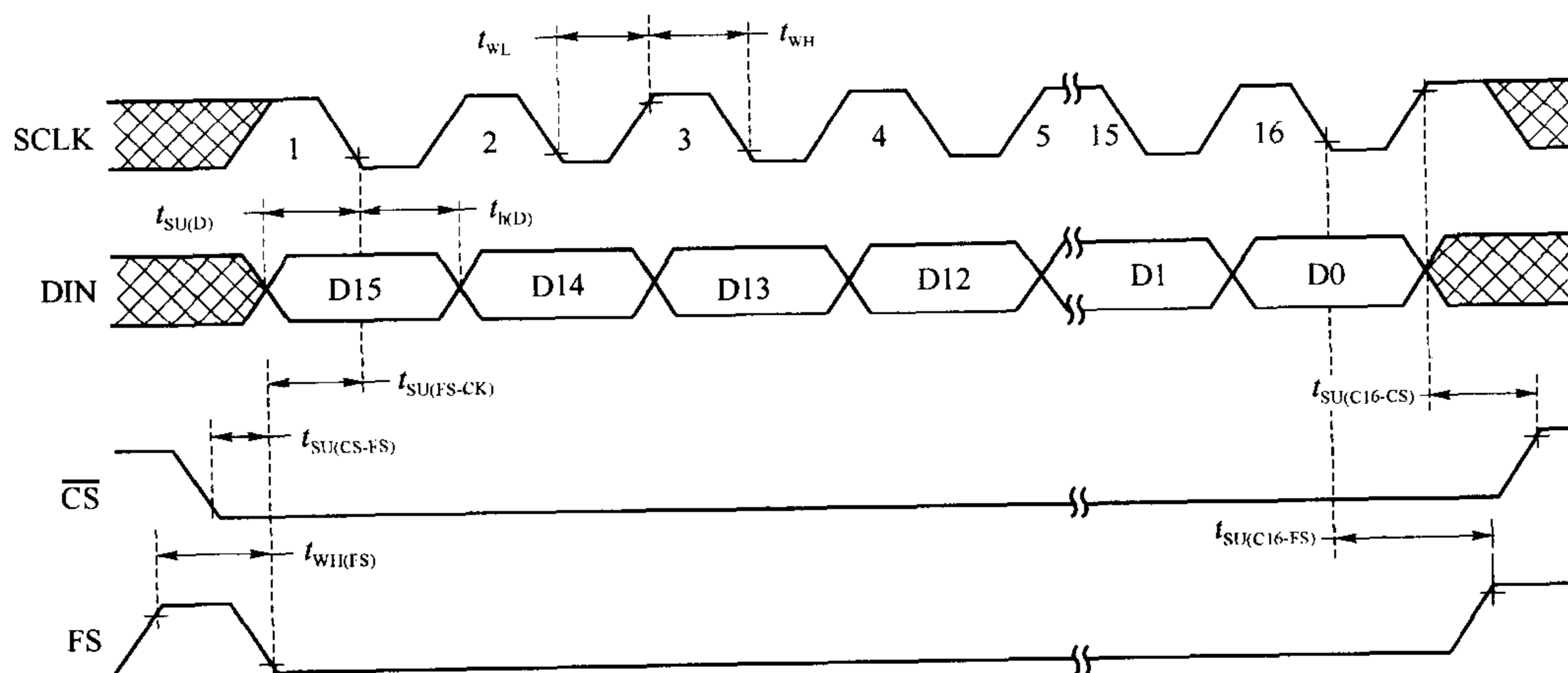


图 10.5.14 TLV5616 的操作时序图

表 10.5.6 图 10.5.14 的对应参数表

符 号	参 数	说 明	最小值	单 位
$t_{\text{SU(CS-FS)}}$	建立时间	$\overline{\text{CS}}$ 低电平到 FS 下降沿时间	10	ns
$t_{\text{SU(FS-CK)}}$	建立时间	FS 低电平到 SCLK 第一个下降沿时间	8	ns
$t_{\text{SU(C16-FS)}}$	建立时间	FS 置低电平后, SCLK 的第 16 个下降沿到 FS 上升沿的时间(期间采样 D0 位)	10	ns
$t_{\text{SU(C16-CS)}}$	建立时间	SCLK 的第 16 个上升沿(即采样 D0 后的第一个上升沿)到 $\overline{\text{CS}}$ 上升沿的时间。如果利用 FS 代替 SCLK 的第 16 个上升沿更新 DAC, 则建立时间为 FS 上升沿到 $\overline{\text{CS}}$ 上升沿的时间	10	ns
$t_{\text{WH}}$	脉冲持续时间	SCLK 高电平持续时间	25	ns
$t_{\text{WL}}$	脉冲持续时间	SCLK 低电平持续时间	25	ns
$t_{\text{SU(D)}}$	建立时间	SCLK 下降沿之前的数据准备就绪时间	8	ns
$t_{\text{h(D)}}$	保持时间	SCLK 下降沿之后的数据有效时间	5	ms
$t_{\text{WH(FS)}}$	脉冲持续时间	FS 高电平持续时间	20	$\mu\text{s}$

### 5. AT89S52 和 TLV5616 的接口与编程

TLV5616 与 AT89S52 单片机的接口电路如图 10.5.15 所示。

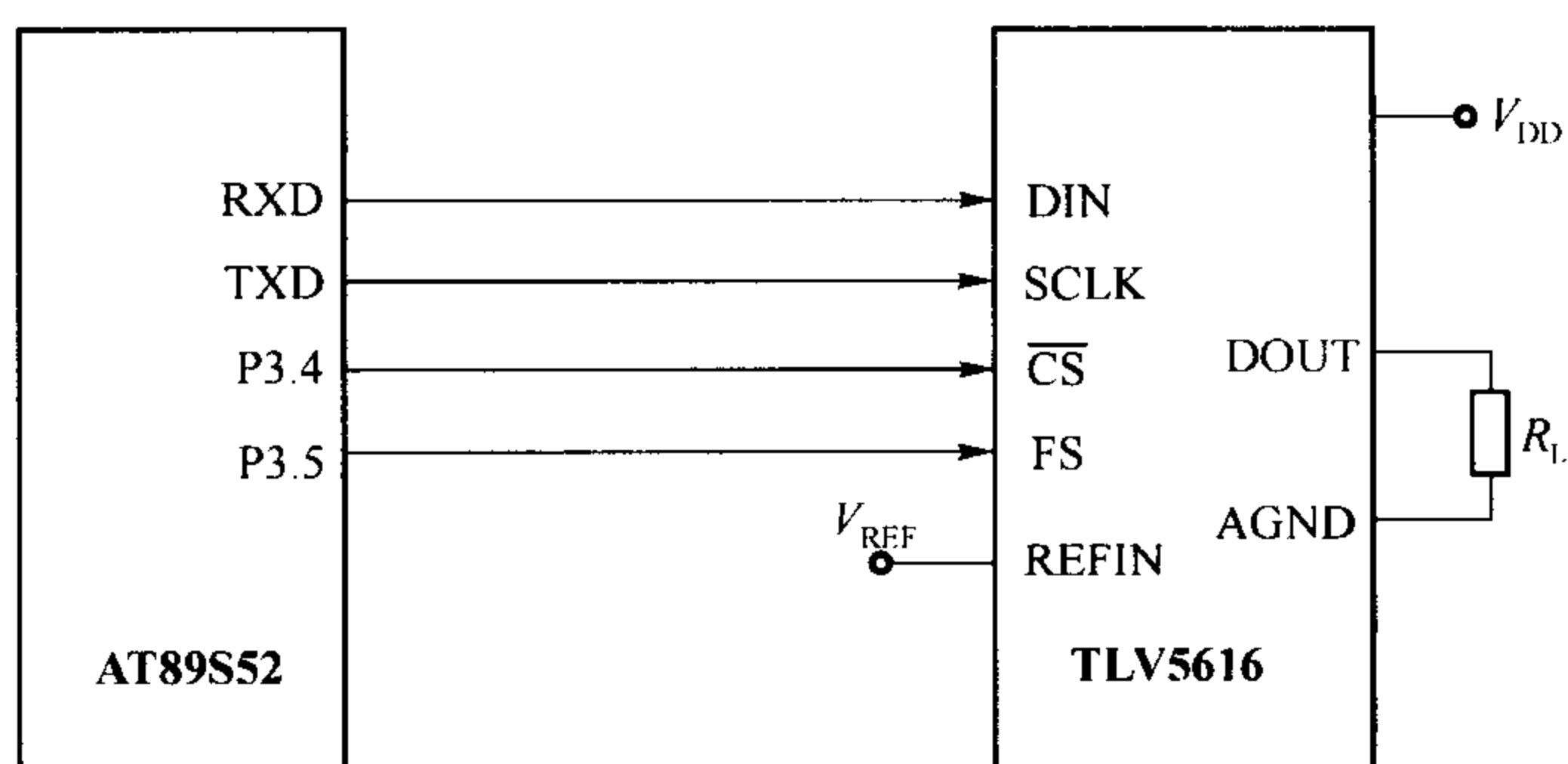


图 10.5.15 TLV5616 和 AT89S52 接口电路

单片机的晶振频率为 12 MHz, RXD 接 DIN, 串行输出待转换数据; TXD 产生串行时钟接 SCLK, 单片机工作于方式 0, P3.4 产生片选信号; P3.5 产生帧同步信号 FS; 参考电压可根据要求连接, 转换后的波形从 DOUT 以电压的形式输出, T0 每 60 ms 中断一次, 读入一个数据进行转换。在数据格式中, 选择 SPD=0, PWR=0, 将待转换的数据存放于数据表中, 执行下列程序将表中数据转换成模拟信号, 从 DOUT 输出。

```

ORG    0000H
LJMP   START
ORG    000BH
LJMP   TIMER0
ORG    0100H

```

```

START:  MOV    SP, #60H    ;置堆栈指针
        CLR    A
        MOV    SCON, A    ;串行口工作于方式 0
        MOV    TMOD, #02  ;T0 工作于方式 2
        MOV    TH0, #0C8H ;置定时器初值
        MOV    TL0, #0C8H
        SETB   P3.5       ;置 FS = 1
        SETB   P3.4       ;置  $\overline{CS}$  = 1
        SETB   ET0
        SETB   EA
        MOV    R0, A       ;地址指针清 0
        SETB   TR0
ALWAYS: JMP    ALWAYS     ;等待中断
        ORG    1000H

TIMER0: PUSH   PSW        ;T0 中断服务
        CLR    P3.4       ; $\overline{CS}$ 变低
        CLR    P3.5       ;FS 变低
        MOV    DPTR, #VALS ;置待转换数据表的表头地址
        MOV    A, R0
        MOVC   A, @A + DPTR ;取高位数据
        MOV    SBUF, A     ;发高位数据
        MOV    A, R0       ;取地址指针
        INC    A           ;地址加 1
        MOVC   A, @A + DPTR ;取低位数据
MSB-TX: JNB    TI, MSB - TX ;高字节数据发完
        CLR    TI
        MOV    SBUF, A     ;查低字节数据
LSB-TX: JNB    TI, LSB - TX ;低字节数据发完
        SETB   P3.5       ;置 FS = 1
        CLR    TI
        MOV    A, R0       ;修改地址指针
        INC    A
        INC    A
        MOV    R0, A
        SETB   P3.4       ;置  $\overline{CS}$  为高
        POP    ACC

```

```

        POP     PSW
        RETI
    VALS:
        :
    END

```

## 10.6 实时时钟电路 DS1302 的扩展

常用的串行时钟芯片如 DS1302, DS1307, PCF8485 (Philip 公司), 并行时钟芯片如 DS12887 等。DS1302 是美国 Dallas 公司推出的一种高性能、低功耗、带 RAM 的实时时钟芯片。DS1302 是 DS1202 的升级产品, 与 DS1202 兼容, 但增加了主电源/备用电源双电源引脚, 同时提供了对备用电源进行涓细电流充电的能力。

### 1. DS1302 的主要特点

- 工作电压为 2.5~5.5 V。
- 可对年、月、日、星期、时、分、秒进行计时, 闰年补偿, 有效至 2100 年。
- 可采用 12 h 或 24 h 方式计时。
- 可采用突发方式一次传送多个字节的时钟信号或 RAM 数据。
- 内部有一个 31×8 位的用于临时存放数据的 RAM 寄存器。
- 采用双电源(主电源和备用电源)供电, 可设置备用电源充电方式, 并且可以关闭充电功能, 同时提供了对备用电源进行涓细电流充电的能力。
- 采用 SPI 三线接口与 CPU 进行通信。
- 采用普通 32.768 kHz 晶振。

### 2. DS1302 的引脚及功能

8 脚 DIP 封装的引脚排列如图 10.6.1 所示。

$V_{CC1}$  接后备电源,  $V_{CC2}$  接主电源: 在主电源关闭的情况下, 也能保持时钟的连续运行。DS1302 由  $V_{CC1}$  或  $V_{CC2}$  两者中的较大者供电。

当  $V_{CC2} > V_{CC1} + 0.2 \text{ V}$  时,  $V_{CC2}$  给 DS1302 供电。当  $V_{CC2} < V_{CC1}$  时, DS1302 由  $V_{CC1}$  供电。

X1 和 X2: 外接 32.768 kHz 晶振, 为芯片提供计时脉冲。

$\overline{\text{RST}}$ : 复位/片选线。

I/O: 串行数据输入输出端(双向)。

SCLK: 串行时钟输入端。

### 3. DS1302 的控制字及寄存器

通过对 DS1302 寄存器的控制字进行读写操作完成对 DS1302 的读写。

#### (1) DS1302 的控制字

DS1302 的控制字如图 10.6.2 所示。控制字的最高有效位(位 7)必须是逻辑 1, 如果

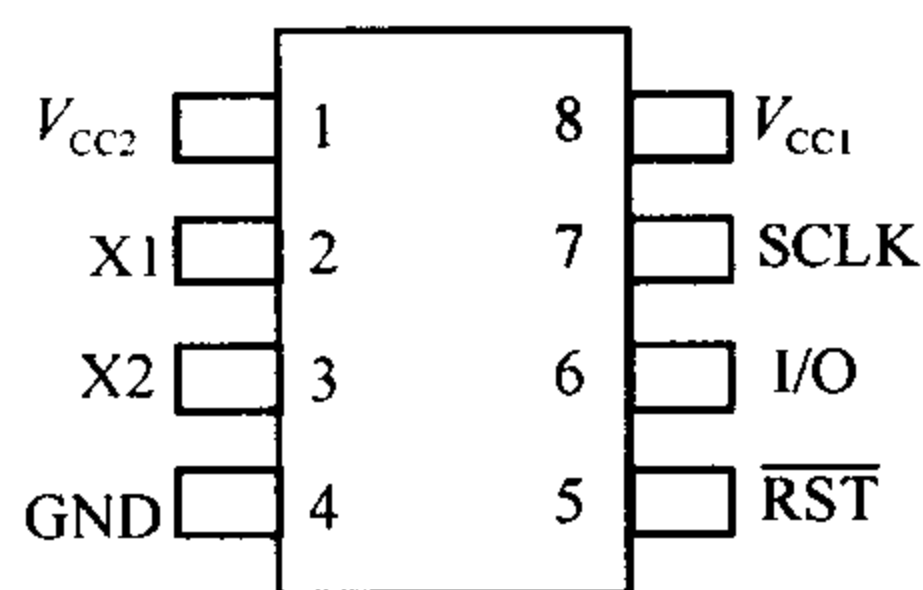


图 10.6.1 DS1302 的引脚图

1	RAM CK	A4	A3	A2	A1	A0	R/W
---	-----------	----	----	----	----	----	-----

在 DS1302 中,通过把  $\overline{\text{RST}}$  置高电平以启动所有的数据传送。 $\overline{\text{RST}}$  有两种功能:首先, $\overline{\text{RST}}$  接通控制逻辑,允许地址/命令序列送入移位寄存器;其次,利用 RST 可终止单字节或多字节数据的传送。当 RST 为高电平时,所有的数据传送被初始化,允许对 DS1302 进行操作。如果在传送过程中置 RST 为低电平,则会终止此次数据传送,并且使 I/O 引脚变为高阻态。上电运行时,在  $V_{\text{CC}} \geq 2.5 \text{ V}$  之前, $\overline{\text{RST}}$  必须保持低电平。只有在 SCLK 为低电平时,才能将 RST 置为高电平。

突发方式

DS1302 有 12 个寄存器,其中有 7 个寄存器与日历、时钟相关,数据格式为 BCD 码,其日历、时间寄存器及其控制字如表 10.6.1 所示。控制寄存器及其控制字如图 10.6.4 所示,WP=1 为写保护。

表 10.6.1 日历、时间寄存器及其控制字

寄存器名称	命令字		取值范围	各位内容							
	写操作	读操作		7	6	5	4	3	2	1	0
秒寄存器	80H	81H	00~59	CH	10SEC			SEC			
分寄存器	82H	83H	00~59	0	10MIN			MIN			
时寄存器	84H	85H	01~12 或 00~23	12/24	0	10	HR	HR			
日寄存器	86H	87H	01~28, 29, 30, 31	0	0	10DATE		DATE			
月寄存器	88H	89H	01~12	0	0	0	10M	MONTH			
周寄存器	8AH	8BH	01~07	0	0	0	0	0	DAY		
年寄存器	8CH	8DH	00~99	10YEAR				YEAR			



图 10.6.4 控制寄存器及控制字

此外,DS1302 还有年份寄存器、控制寄存器、充电寄存器、时钟突发寄存器及与 RAM 相关的寄存器等。时钟突发寄存器可一次性顺序读写除充电寄存器外的所有寄存器内容。DS1302 与 RAM 相关的寄存器分为两类:一类是单个 RAM 单元,共 31 个,每个单元组态为一个 8 位的字节,其命令控制字为 C0H~FDH,其中奇数为读操作,偶数为写操作;另一类为突发方式下的 RAM 寄存器,此方式下可一次性读写所有的 RAM 的 31 B,命令控制字为 FEH(写)、FFH(读)。

4. DS1302 与 AT89S52 的接口电路

图 10.6.5 为单片机 AT89S52 与 DS1302 的连接电路图。单片机控制 DS1302 的程序主要包括对寄存器的地址定义和控制字的写入,以及数据的读取。主要包括:初始化 DS1302 主程序,读 DS1302 时间子程序,读字节子程序,写字节子程序。

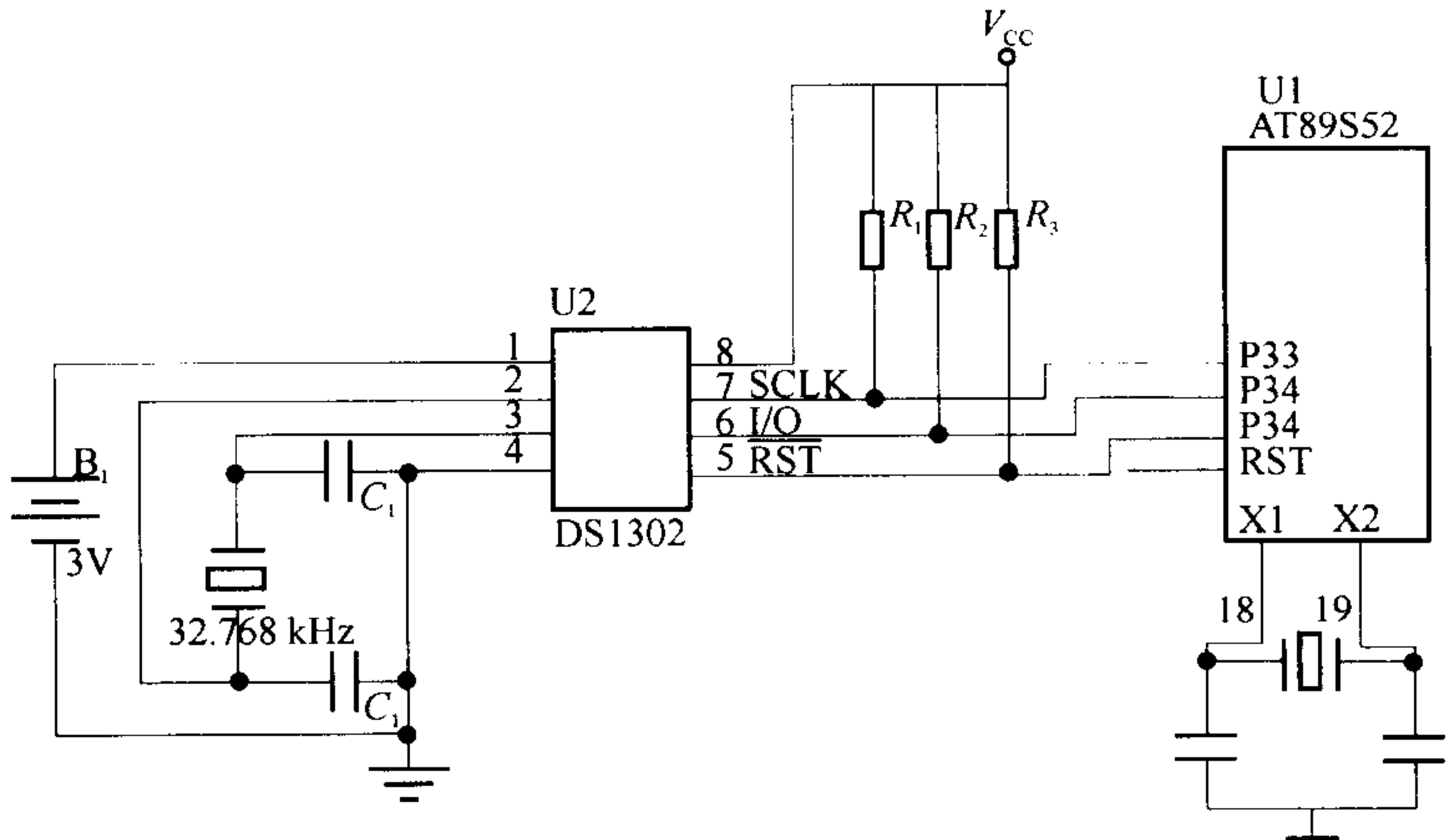


图 10.6.5 DS1302 与 CPU 连接的电路原理图



## (1) 初始化 DS1302 主程序

功能: 设置 DS1302 初始时间, 并启动计时。

初始时间在: Second, Minute, Hour, Day, Month, Week, Year 寄存器(地址连续)

```

                CLK Bit  P3.3          ;实时时钟时钟线引脚
                IO Bit   P3.4          ;实时时钟数据线引脚
                RST Bit  P3.5          ;实时时钟复位线引脚
                org      0000H
                ljmp     main
main:           CLR      RST
                CLR      CLK
                SETB     RST
                MOV      R2, #8EH      ;控制寄存器
                LCALL    wrbyte
                MOV      R2, #00H      ;写操作前 WP = 0
                LCALL    wrbyte
                SETB     CLK
                CLR      RST
                MOV      R0, #Second
                MOV      R7, #7        ;秒分时日月星期年共7个连续字节单元
                MOV      R1, #80H      ;秒写地址
set1:          CLR      RST
                CLR      CLK
                SETB     RST
                MOV      A, R1          ;写秒分时日月星期年地址
                MOV      R2, A
                LCALL    wrbyte
                MOV      A, @R0        ;写秒数据
                MOV      R2, A
                LCALL    wrbyte
                INC      R0
                INC      R1
                INC      R1
                SETB     CLK
                CLR      RST
                DJNZ     R7, set1
                CLR      RST
                CLR      CLK
                SETB     RST

```

```

MOV      R2, #8EH          ;控制寄存器
LCALL    wrbyte
MOV      R2, #80H          ;控制,WP = 1,写保护
LCALL    wrbyte
SETB     CLK
CLR      RST
Rdtime:  LCALL    rd1302
          SJMP     rdtime

```

### (2) 读 DS1302 时间子程序

功能:从 DS1302 读时间;调用 wrbyte,rdbyte 函数。

出口参数:时间保存在 Second,Minute,Hour,Day,Month,Week,Year 寄存器。

```

rd1302:  MOV      R0, #Second
          MOV      R7, #7          ;寄存器个数
          MOV      R1, #81H        ;秒地址
rd13021: CLR      RST
          CLR      CLK
          SETB     RST
          MOV      A, R1            ;秒分时日月星期年地址
          MOV      R2, A
          LCALL    wrbyte
          LCALL    rdbyte
          MOV      @R0, A          ;秒
          INC      R0
          INC      R1
          INC      R1
          SETB     CLK
          CLR      RST
          DJNZ     R7, rd13021
          RET

```

### (3) 写 DS1302 单字节子程序

功能:写 DS1302 单字节数据;入口参数 R2,写数据。

```

wrbyte:  MOV      R4, #8          ;8 位数据
wrbit:   MOV      A, R2            ;取写数据
          RRC      A                ;移位
          MOV      R2, A            ;暂存
          MOV      IO, C            ;位送 I/O 数据线
          SETB     CLK              ;时钟置高电平
          CLR      CLK              ;时钟置低电平

```

```

    DJNZ    R4, wrbit      ;8 位未写完,继续
    RET

```

#### (4) 读 DS1302 单字节子程序

功能:读 DS1302 单字节数据;出口参数 A,读出数据。

```

rdbyte:  MOV    R4, #8      ;读 8 位数据
rdbit:   MOV    C, IO
          RRC     A
          SETB    CLK
          CLR     CLK
          DJNZ    R4, rdbit  ;8 位未读完,继续
          RET

```

## 习 题

1. 如何构建 AT89S52 单片机扩展的系统总线?
2. 什么是线选法? 什么是地址译码选通法? 各有什么特点?
3. 简述单片机系统扩展的基本原则和实现方法?
4. 请设计出采用两片 2764 EPROM 芯片扩展 16 KB 程序存储器的 AT89S52 单片机系统硬件连接图。
5. 8155 有哪几种工作方式? 怎样进行选择?
6. 试编程对 8155 进行初始化,设 A 口为选通输出,B 口为选通输入,C 口作为控制联络口,并启动定时/计数器按方式 1 工作,工作时间为 10 ms,定时器计数脉冲频率为单片机的时钟频率 24 分频, $f_{osc}=12\text{ MHz}$ 。
7. 要求 8255A 的 A 口工作在方式 0 输出,B 口工作在方式 1 输入,C 口的 PC7 为输入,PC1 为输出,试编写 8255A 的初始化程序。
8. 利用 8255 设计一个键盘/LED 显示器电路。
9. 在 AT89S52 单片机系统中接入一片地址为 7FF8H~7FFFH 的 ADC0809 芯片,试画出系统硬件连接图,并编写 ADC0809 初始化程序和定时采样通道 1 的程序。
10. 设计一个 AT89S52 单片机与 AD574 的接口电路,并编写相应的数据采集程序。
11. 设计一个 AT89S52 单片机与 DAC0832 的接口电路,要求采用单缓冲方式,将内部 RAM 20H~2FH 单元的数据变换成模拟电压,每隔 1 ms 输出一个数据。

## 实 践 训 练

学完本章内容后,为掌握系统扩展技术、了解各种扩展的基本方法、掌握编程方法,可按照下述步骤进行实践训练。

1. 数据存储器扩展训练。按图 10.3.5 进行电路连接,编程实现对数据存储器某些单元数据的读写。可以将读出数据利用串行口传输到计算机,以便观察。扩展一片

EEPROM AT24C01 或 X84041,编写数据读写程序。

2. I/O 接口扩展训练。按图 10.4.9 进行电路连接,扩展键盘和显示,BIC8718 可以利用 74LS06 代替,调试电路和程序。

(1) 将 74LS06 改为 74LS07 再调试程序,比较其程序的区别。

(2) 调试时改变 8155 的地址线,再重新调试程序。

3. A/D,D/A 扩展训练,熟悉串行、并行 A/D、D/A 的使用方法。在第 2 题电路的基础上,仿照图 10.5.8 电路,扩展 TLC2543 及 TLV5616 芯片,进行以下调试:

(1) 在 TLC2543 的 AIN0 引脚输入电压,编写程序,A/D 转换结果利用数码管显示;可以利用电位器改变输入电压的数值。

(2) 键盘输入 D/A 转换的数字量,编写程序将键盘输入数字量转换成模拟量输出,利用电压表检测模拟量数值。测试不同的数字量,模拟输出的变化情况。

## 第 11 章 单片机应用系统设计及举例

通过前面各章的学习,已经掌握了 AT89S52 单片机的工作原理、编程调试、外围芯片的扩展等,它们是单片机应用系统设计的硬件及软件基础。具有了这些基础以后,就可以进行单片机应用系统的设计。

单片机应用系统是以单片机为核心器件并扩展一些外围电路和设备的应用系统。由于单片机具有功能强、体积小、易开发、性价比高等特点,在许多领域的自动化、智能化方面得到了广泛应用。单片机应用系统可以用于工业控制、智能化仪器、医疗仪器、家用电器、通信等领域。

### 11.1 单片机应用系统的开发过程

单片机应用系统的开发过程主要包括理论设计和实际调试两个阶段。由于单片机系统应用场合的区别很大,所以实际的开发过程会涉及许多方面的知识,如数据采集系统会涉及到各种各样传感器的应用,控制系统会涉及到采用什么样的控制算法以及执行机构等,所以单片机应用系统的开发必须具备以下几个方面的知识和能力。

#### 1. 必须具备的知识和能力

##### (1) 必须具有一定的硬件基础知识

这些硬件基础知识除了单片机及其外围电路的扩展原理及方法外,还包括对设备或仪器进行信息输入或设定的键盘和开关、检测各种输入量的传感器、控制用的继电器等执行装置、与各种仪器进行通信的接口,以及打印和显示等设备的工作原理、电路连接方法和实际应用中的注意事项等。

##### (2) 需要有一定的动手能力

单片机应用系统的设计必然伴随着元器件的焊接、调试过程。这就要求开发人员能够熟练地应用各种测试仪器,熟悉各种信号的检测方法,并具有较强的分析问题、解决问题的能力。当系统出现故障时,能够及时定位故障,分析、推理故障产生的原因,并找到合适、合理、全面的故障解决办法。

##### (3) 需要具备一定的软件设计能力

熟悉软件开发设计思想,较好地规划、组织软件的结构,按照模块化的设计方法,从顶向下,逐步求精。有些复杂的程序,应该绘制清晰的程序总体框图,以及各部分的程序流程图,根据系统要求,灵活地设计出所需要的程序,并在程序中适当考虑系统的可扩展功能。

##### (4) 具有综合运用新知识和新技术的能力

时代不断前进,知识和技术在不断地更新。当旧的知识和技术不能适应发展的需要时,新的知识和技术便应运而生。所以,一个优秀的设计开发人员应该紧跟时代前进的步伐,及时更新所学知识,及时接受、掌握、应用新知识、新技术。

### (5) 搜集、检索、提炼有用知识和资料的能力

能够综合运用所学的知识,做到举一反三。能够较好地利用各种书籍、互联网等工具搜集所需要的资料,从中检索、提炼出有用的部分并将其应用于系统设计,以提高系统开发的质量和效率。设计时尽量借鉴已有的经验、成果以及成熟的技术,在这些基础上,再根据具体要求反复推敲、更新设计方案,并确定最终合理的设计方案。

### (6) 必须了解生产工艺或制造工艺

不管设计开发什么样的系统,设计开发人员必须熟悉工艺流程或制造工艺,了解工艺控制的参数,根据工艺确定测量的参数、范围、精度、控制方法、控制顺序等。如药品生产线发酵环节温度的控制范围、医学手术系统指令控制的顺序、工业现场阀门开启关闭的顺序等都必须根据工艺和实际要求确定。这是开发设计符合实际要求、控制正确、功能完善系统的基础和保证。

## 2. 单片机应用系统开发的步骤

### (1) 系统的目标任务

开发设计一个单片机应用系统或者设计一种智能化的仪器,首先要明白做什么,然后才是怎么做。目标任务即系统要求实现的功能以及技术指标。应用的场合不同,具体的要求会有区别。这些目标任务的提出一般由开发系统的投资方提出,开发设计人员认可。如开发一套单片机路灯控制系统,首先要明确功能要求,如:定时开灯、关灯,根据季节的变化改变开灯、关灯时间,故障路灯的状态信息及时反馈、某些路灯的单独控制以及成本信息等。目标任务要尽可能清晰、完善,完整的目标任务为后续系统的设计和开发奠定坚实的基础。有些目标任务在开始设计时并不是非常清楚、完善,随着系统的研制开发、现场的应用以及市场的变化可能会有不断的更新和变化,设计方案要尽可能适应这些变化。

### (2) 系统的总体设计

根据上一步的功能以及技术指标要求,确定系统的总体设计方案。系统的总体设计包括单片机的选择、重要环节关键器件的选型、技术指标的实现、硬件软件功能的划分等。在此阶段要对元器件市场情况有所了解。

单片机以及关键器件的选择一定要考虑技术是否成熟,是否满足系统的精度、速度和可靠性要求,货源是否充足等,如出现问题是否有可以替代的器件等。技术指标与整个系统的硬件与软件都有关系,所以要综合考虑,硬件选择满足精度要求的产品,软件采用合适的数学模型和算法。硬件、软件功能在一定程度上具有互换性,即有些硬件电路的功能可用软件实现,反之亦然。具体采用什么方法,要根据具体要求及整个系统的性能价格比,加以综合平衡后确定。一般而言,使用硬件完成速度较快,可节省 CPU 的时间,但价格相对昂贵,而且系统比较复杂,势必增加硬件设计和调试的工作量和难度。用软件实现则相对经济,但占用 CPU 较多的时间。所以一般的原则是:在 CPU 时间允许的情况下,尽量采用软件。所以总体设计时,必须权衡利弊,仔细划分硬件和软件的功能。

总体方案决定整个系统的硬件和软件设计,其质量的好坏直接影响整个设计开发过程,所以系统总体设计一定要从系统的目标任务入手,慎而又慎、多方验证、细节与整体统筹考虑,尽可能多地参考国内外同类产品的有关资料,综合考虑系统的可靠性、先进性、通用性、可维护性、可扩展性和成本等,使确定的技术方案合理且符合有关标准。

### (3) 系统的结构框图

系统的总体结构设计完成后,将整个系统划分成若干模块,利用框图表示出各模块之间的关系、数据流向、控制流向,说明各模块的工作原理、采用的核心技术以及实现的功能。结构框图将整个系统的结构图形化、清晰化、简单化,有助于对系统的进一步理解和掌握,并为硬件和软件设计的模块化打下基础。

### (4) 系统的硬件设计

系统的硬件设计是根据总体设计方案以及结构框图,在所选择的单片机以及关键元器件的基础上,再进一步确定系统中所要使用的元器件,分模块绘制系统的电路原理图,最后将各模块的电路图综合起来,得到系统的总体电路图,并依据电路图设计、制作印刷电路板以及组装样机等。设计时要综合考虑各元件的驱动和带负载能力,要根据情况进行扩展,必要时做一些部件环节实验以验证电路的正确性。电路板设计时要综合考虑模拟电路、数字电路;高频电路、低频电路;高压电路、低压电路的布线规则,地线的布线方法和原则,以及印刷电路板导线宽度与所能承受的电压、电流关系等,并要综合考虑抗干扰设计。

### (5) 系统的软件设计

根据系统总体设计方案中软件实现的功能,明确数学模型和算法,遵循自顶向下、模块化设计的原则,综合顺序程序设计、分支程序设计、子程序设计、中断服务程序设计的各种方法,绘制程序流程图,并编写相应的程序。

### (6) 系统的联机调试、运行和维护

系统的联机调试指对整个系统的硬件和软件进行整体调试,是个比较复杂、难度相当大的过程,调试的方法因人而异。一般有以下原则,不管硬件调试还是软件调试都要采用个个击破的方法,具体就是分模块进行,而且大的模块又可以分成小的模块。比如单片机模块,首先要检查单片机是否正常复位、是否起振,因为这是它正常工作的前提,然后才能验证程序是否正常运行。硬件是软件的工作平台,软件只有工作在正确无误的硬件平台上才能验证其正确性,所以一般的调试过程是硬件调试成功后再调试软件。软件利用开发系统先进行模拟仿真后,再进行在线仿真调试。整个系统联机调试成功后,需要先在实验环境运行,认真仔细地记录其运行状态、故障状态、连续运行时间等,最后写出书面报告,根据运行报告再进行相应的硬件或软件改动。实验环境运行满足要求后,还要在现场环境运行,现场环境相对实验环境要复杂得多,一定要认真观察运行情况,分析出现的各种故障及原因,此时出现故障时,尽量采用软件的方法修正。

系统在实际工作过程中,可能会受到来自系统内部和外部的各种各样的干扰,使系统发生异常状态。通常把瞬时的不加修理也能恢复正常的异常状态称为错误;而必须通过修理才能恢复正常的异常状态称为故障。

### (7) 可靠性设计

对于单片机应用系统来说,可靠性是最重要、最基本的技术指标。如果一个单片机应用系统不能保证稳定可靠地工作,则其他的功能无从谈起。单片机应用系统的可靠性指在规定的条件下和规定的时间内,完成规定功能的能力。规定的条件包括环境条件(如温度、湿度、振动、电磁干扰等)、使用条件、维修条件、操作水平等。常用的描述可靠性的定

量指标有可靠度、失效率、平均无故障时间。

可靠度指产品或系统在规定条件下和规定的时间内完成规定功能的概率。

失效率又称故障率,指工作到某一时刻尚未失效的产品在该时刻后单位时间内发生失效的概率。

平均寿命又称平均无故障工作时间,指产品寿命的平均值。

为了减少系统的错误和故障,系统设计时常从以下几个方面提高系统可靠性:冗余设计,电磁兼容设计,信息冗余技术,时间冗余技术,故障自动检测与诊断技术,软件可靠性技术,失效保险技术等。

所以对于一个实际应用系统首先要保证可靠,其次是实时,然后是灵活和通用。

## 11.2 液氧容器温度控制系统设计

液氧容器温度控制系统应用于液氧容器生产线。液氧容器生产线如图 11.2.1 所示。每条生产线上放置 40 个液氧容器,液氧容器由不锈钢材料制作,圆筒状、两层、中空结构,用于存放液氧以备急需或高山应用。液氧容器为了适应高山环境,需要经过抽真空及加温处理。加温处理对液氧容器的质量和合格率有重要的影响,所以要求严格控制温度的范围,实现自动控制液氧容器的处理温度,提高产品的质量和合格率。

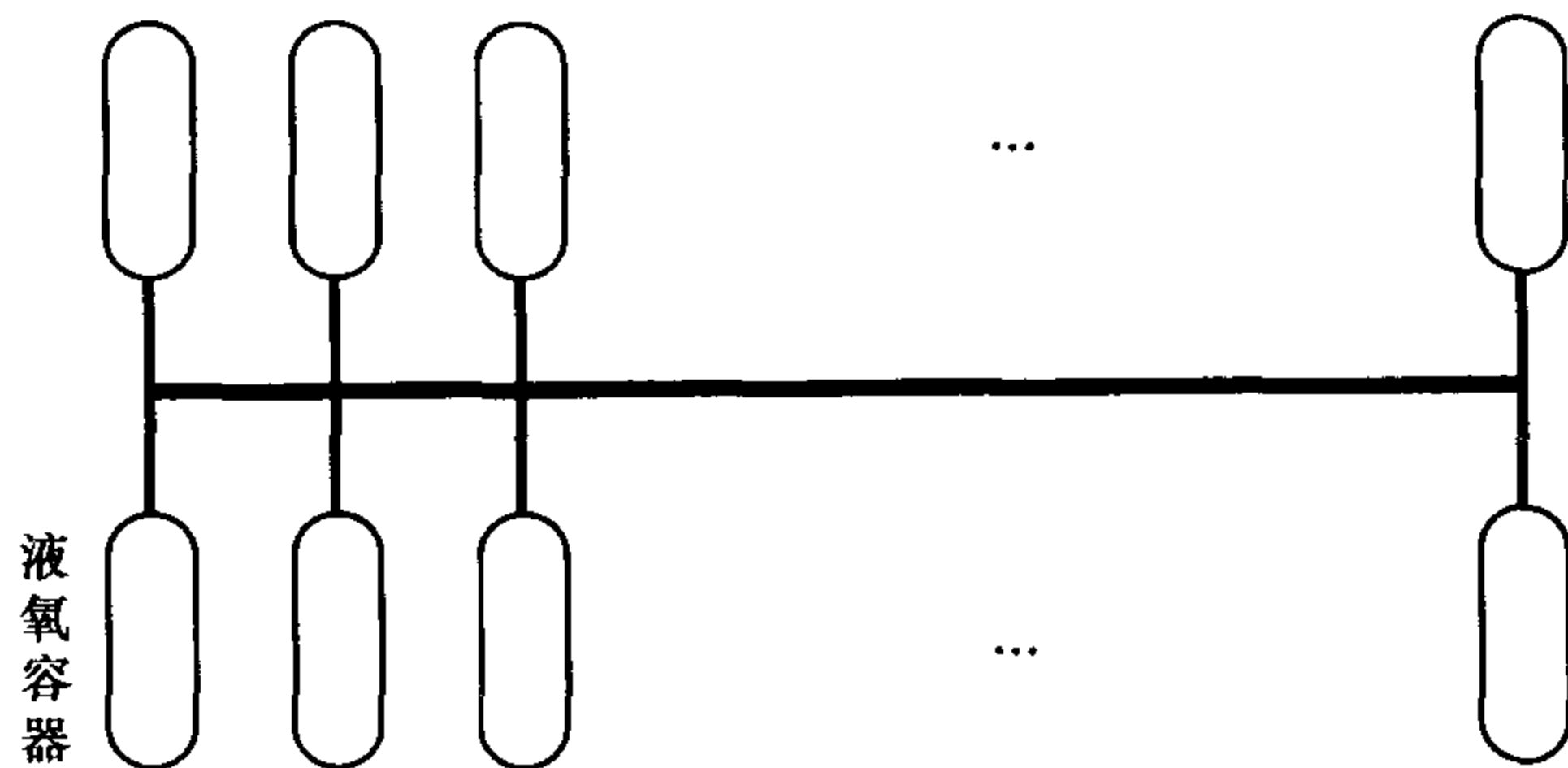


图 11.2.1 液氧容器生产线结构

### 11.2.1 系统的目标任务

目标任务及现场条件如下:

- 主控室计算机可以对液氧容器的温度进行实时采集和控制,并绘制每点的温控曲线;
- 温度控制范围:30~120 °C;
- 温度控制精度:±0.5 °C;
- 主控室距离控制现场最近的液氧容器 15 m,最远的液氧容器 100 m;
- 要求系统工作稳定、可靠,控制准确。

### 11.2.2 系统的总体设计

根据以上目标任务,为了保证系统工作稳定可靠,采用集散控制的方法,每两个液氧



容器配备一台现场控制器,现场控制器只有在接收到主控计算机的指令时才与其通信,其他时间单独工作,其特点如下:

- 当计算机系统出现故障时,现场控制器可以继续工作,不会影响控制功能;
- 温度控制器有 3 种工作状态,导通、截止、控制,具体工作方式由计算机设置;
- 当某一台现场控制器出现故障时,可以立即利用备用的现场控制器替换,不会影响其他液氧容器的控制,保证控制质量。

本系统的设计包括现场控制器的硬件、软件设计,计算机监控程序设计。按照目标要求,现场控制器完成的功能如下:

- 采集现场温度并传输给计算机,根据设定温度进行实时控制,满足控制要求;
- 与计算机进行通信,并按照命令进行相应的操作;
- 显示当前的温度测量值、设定值、控制参数;
- 键盘可以输入设置参数以及温度的设置值。

计算机软件完成的功能如下:

- 采集测量温度值,绘制、打印实时曲线;
- 进行上、下限温度报警;
- 与现场控制器进行通信,显示控制器状态。

由于主控室与现场控制器之间的最长距离为 100 m,使用 RS232 总线不能满足要求,所以计算机与现场控制器的通信采用 RS485 串行总线方式。控温范围为 30~120 °C,采用数字温度传感器 DS18B20。由于现场环境有干扰,为了保证系统可靠工作,采用自带看门狗的单片机 AT89S52,保证有干扰时,程序能够自动复位并开始正常工作。

### 11.2.3 系统的结构框图及工作原理

根据系统的目标任务及总体设计方案,绘制系统的总体框图如图 11.2.2 所示。

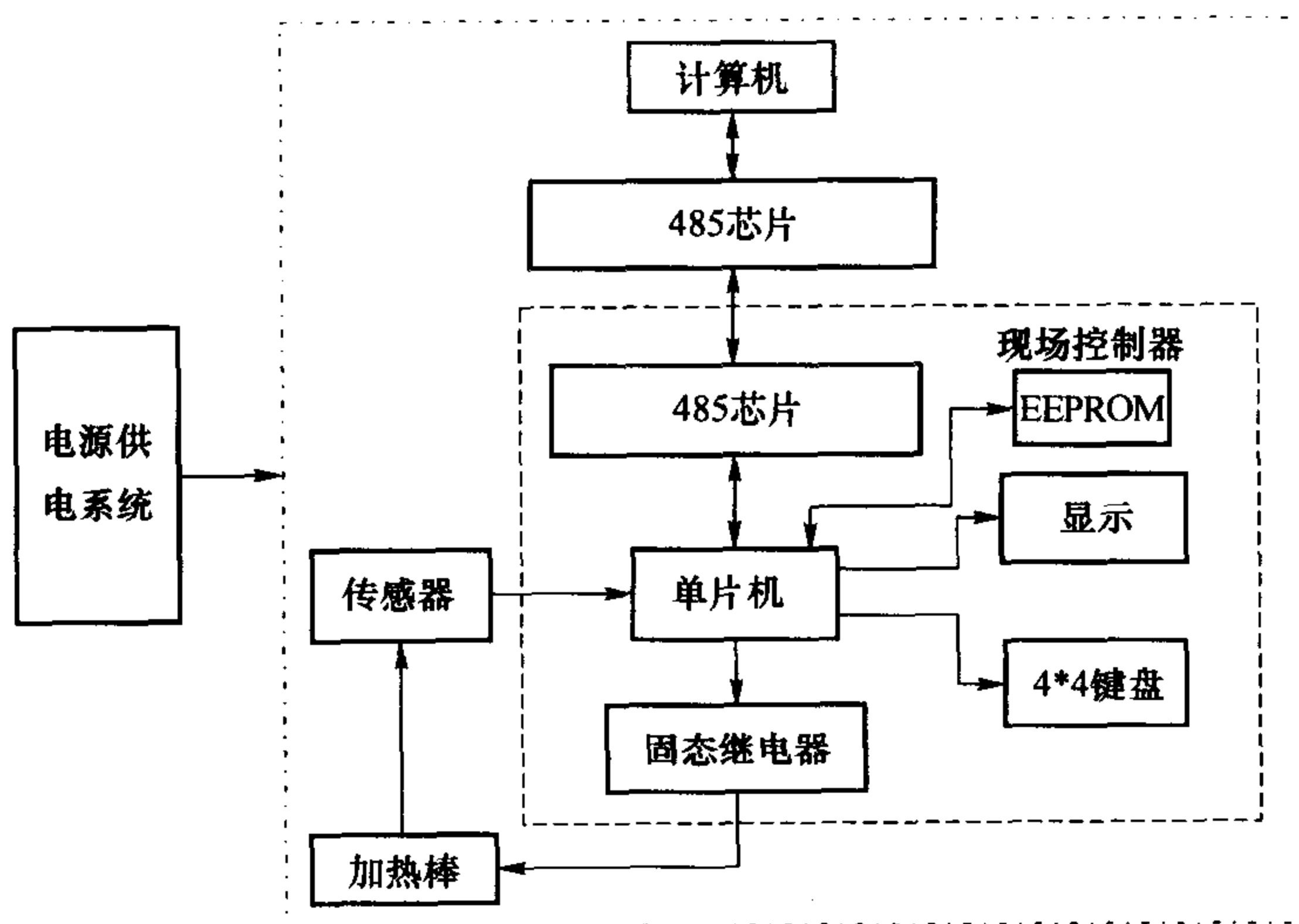


图 11.2.2 系统的总体组成框图

本系统主要由电源供电系统、计算机、RS232 至 RS485 转换器、现场控制器、温度传感器和加热棒组成,各部分功能如下。

(1) 计算机:用于运行监控、显示等程序。显示器采用触摸屏,方便现场工人操作。触摸屏采用压阻式,4 线连接方式,可以在计算机上对各个工位进行分别设置和控制。

(2) 现场控制器:用于采集、显示、控制现场温度,两个工位一台现场控制器。将其温度信号数据上传给计算机。现场控制器与上位机间的通信方式为 RS485 半双工串行通信,利用 485 通信芯片实现。

(3) RS232 至 RS485 转换器:用于完成 RS232 信号到 RS485 信号的双向转换。

(4) 加热棒:每个工位一个加热棒,用于液氧容器的加热。

(5) 温度传感器:每个工位一个传感器,用于检测现场温度信号。

(6) 电源供电系统:提供计算机、现场温度控制器以及加热棒的供电电源。

工作原理:现场温度采集控制器通过传感器采集现场温度,根据输入的控制参数和一定的算法计算输出值,传输给固态继电器实现对加热棒的温度控制。现场温度采集控制器与计算机之间通过 RS485 总线通信。现场控制器有两种数据输入的方法,一是通过键盘,二是通过计算机通信传输信息。数码管用于显示温度以及各种状态信息。

#### 11.2.4 硬件设计

硬件设计主要是现场控制器的设计,由图 11.2.2 可以看出,现场温度控制器由传感器输入、键盘输入、固态继电器输出控制、显示输出、EEPROM 数据存储、通信 6 个模块组成。现场温度采集控制器硬件原理图如图 11.2.3 所示。设计时首先选择好关键元器件,然后再从这 6 个模块入手,逐一设计,最后再统一规划整理。

本系统的关键器件:单片机、温度传感器、固态继电器。由上知单片机及温度传感器已经选定 AT89S52 和 DS18B20,接下来确定固态继电器的型号。

固态继电器在系统中属于关键器件,其稳定、可靠是实现控制的前提。选择步骤如下。

(1) 根据现场情况确定工作时固态继电器的电压、电流。固态继电器的电压、电流根据负载的工作电压、电流确定。如加热棒供电电源为 36 V、1 A,则固态继电器的正常工作电压、电流为 36 V、1 A。

(2) 确定固态继电器的额定电压或电流。额定电压、电流一般取工作电压的 2~5 倍,具体情况酌情选择。本系统取两倍的工作电压,即大于 72 V、2 A。

(3) 确定固态继电器的类型。固态继电器有直流固态继电器和交流固态继电器,根据控制电压是直流或交流确定。本系统加热棒上控制电压为直流,所以选择直流固态继电器。

(4) 确定输入控制信号。由于单片机的输出信号(即固态继电器的输入控制信号)控制固态继电器,而单片机的供电电压在 4~5.5 V,所以固态继电器输入控制信号为 5 V 信号。

(5) 根据上述步骤确定固态继电器指标如下:直流固态继电器,额定电压大于 72 V,输入控制信号 5 V。

(6) 通过网络、书刊等渠道搜集固态继电器生产厂家的资料,从中寻找满足要求的产



品,再根据价格要求、货源情况、技术服务、熟悉程度等最终确定固态继电器型号。

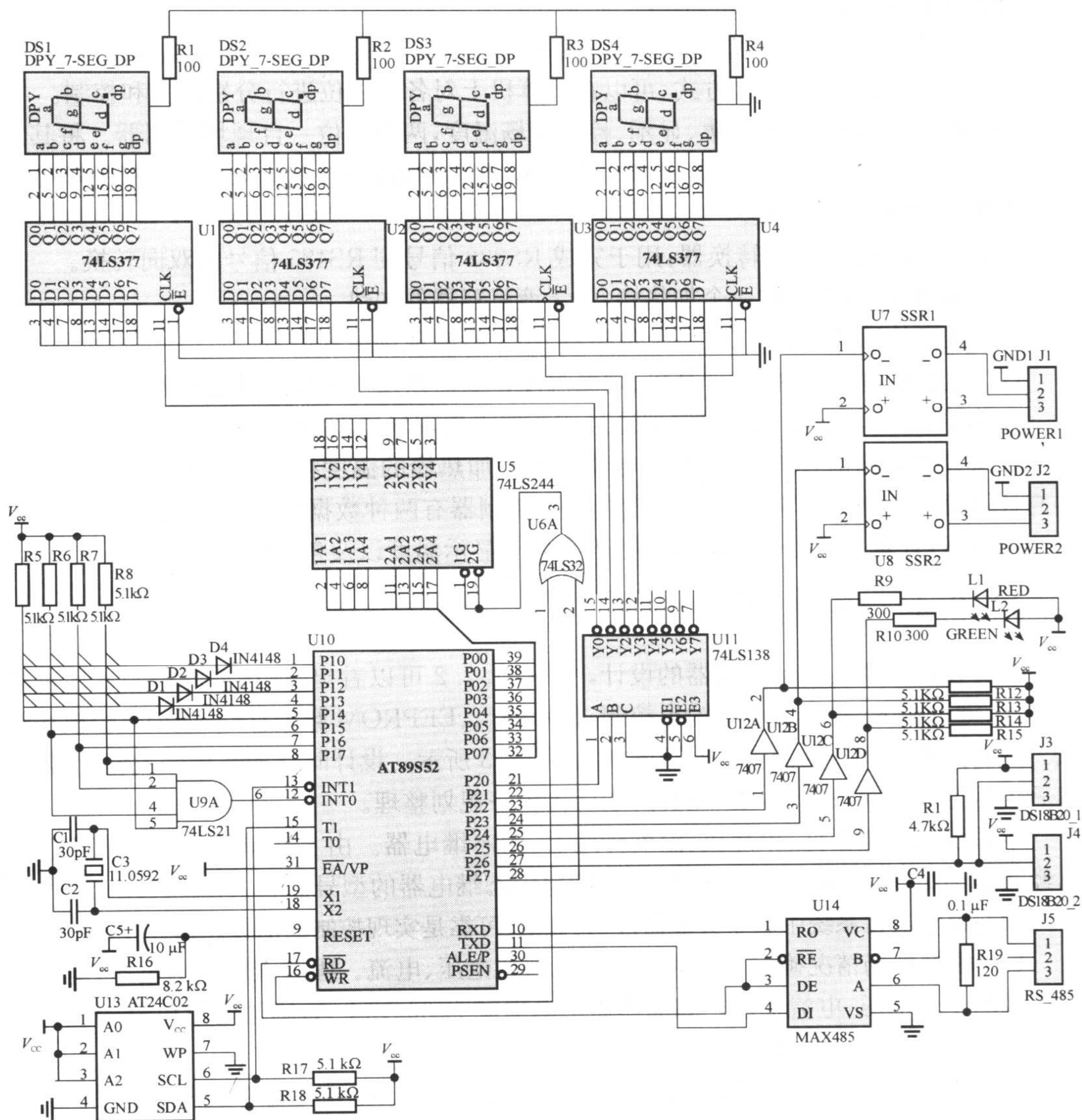


图 11.2.3 现场温度采集控制器硬件原理图

根据以上步骤,固态继电器选择北京科通的 JGX-3A 型号固态继电器,其指标如下:直流固态继电器,输入控制信号 3~36 V,额定电压 80 V,额定电流 3 A。

以上确定固态继电器的步骤也适用于其他元器件的确定。

各元器件的型号确定下来之后,分别进行 6 个模块的硬件设计。为了确保设计准确无误,一些关键器件和不太熟悉的器件应该首先寻找其数据资料和应用资料,以便参考和借鉴。

### 11.2.5 软件设计

根据以上设计,系统的软件由主程序模块、初始化模块、键盘输入模块、温度采集模块、显示模块、算法计算数据处理模块、固态继电器控制模块、EEPROM 读写模块、通信模块、看门狗模块等组成。程序采用 C51 进行编程。主程序流程图如图 11.2.4 所示,其中 NUMBER 与 WDT 有关,要小于 WDT 的溢出数据 8191(1FFFH)个机器周期。

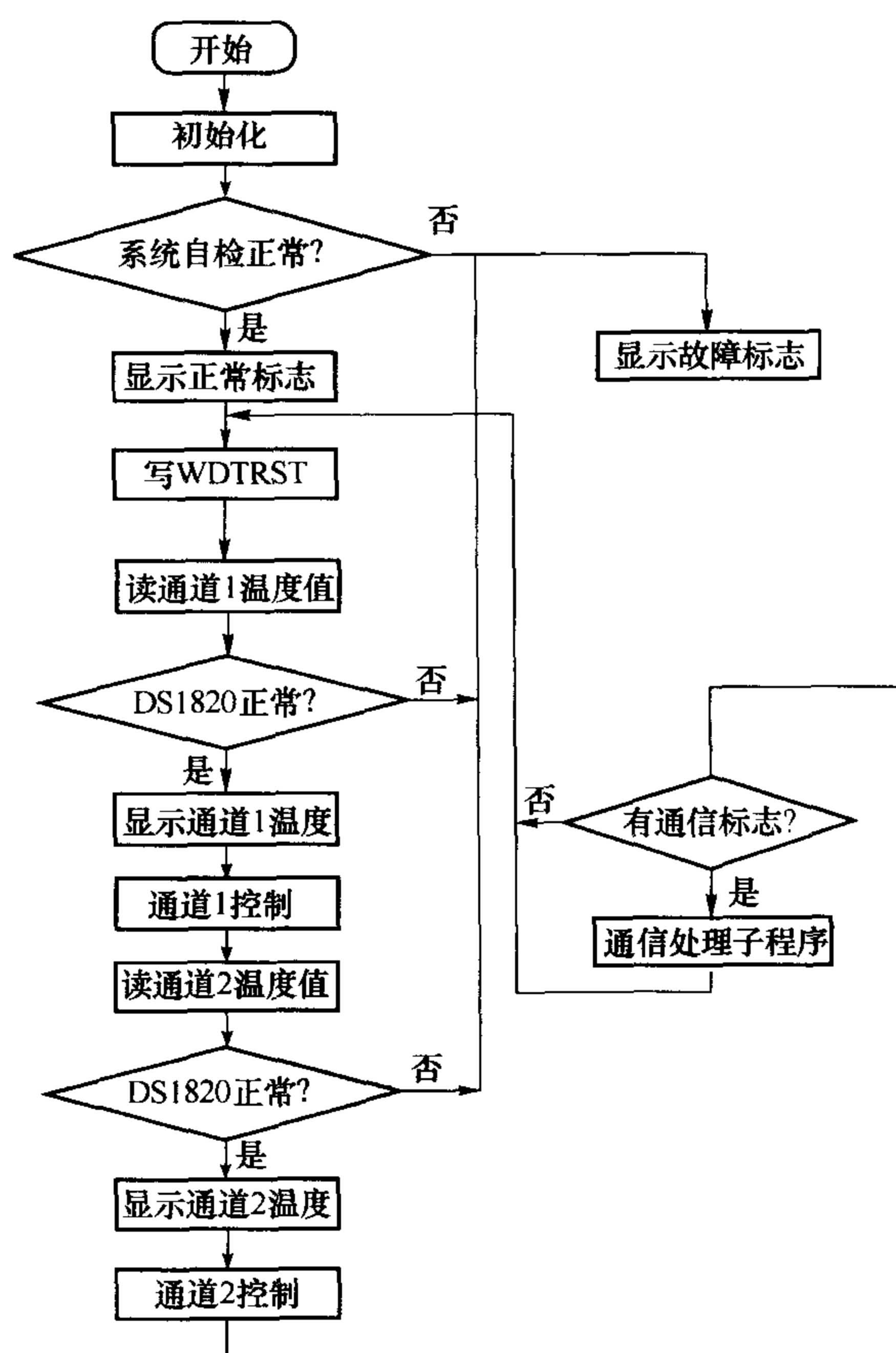


图 11.2.4 主程序流程图

下面主要介绍通信协议、变量定义、主程序、继电器控制模块的程序。

#### 1. 串口通信协议规定如下

(1) 通信测试并取得控制器地址(Communication Test)

格式:@<CR>

应答:!(addr)(#)

addr 格式:0001H~FFFFH 4 位十六进制数,为每台现场控制器的地址。

## (2) 读入温度数据(Read Analog Data)

格式: \$(addr)(R)(1)&lt;CR&gt; 读第一通道

格式: \$(addr)(R)(2)&lt;CR&gt; 读第二通道

应答: &gt;(addr)(2)(1)(Temperature Data)(#), 其中 2 表示第二条协议, 以下同。

&gt;(addr)(2)(2)(Temperature Data)(#)

如: &gt;(addr)(2)(1)(12(高 8 位) 34(低 8 位))(#)

## (3) 设置模块的控制温度(write temp)

格式: \$(addr)(W)(set temp)(1)&lt;CR&gt; 第一路设置温度值

\$ (addr)(W)(set temp)(2)&lt;CR&gt; 第二路设置温度值

应答: &gt;(addr)(3)(1)(#)

&gt;(addr)(3)(2)(#)

## (4) 主机控制加热棒通断和解除控制命令(Hot)

通道 1 格式:

\$ (addr)(H)(1)(1)&lt;CR&gt; //通状态

\$ (addr)(H)(1)(2)&lt;CR&gt; //断状态

\$ (addr)(H)(1)(0)&lt;CR&gt; //解除控制状态

应答: &gt;(addr)(1)(4)

通道 2 格式:

\$ (addr)(H)(2)(1)&lt;CR&gt; //通状态

\$ (addr)(H)(2)(2)&lt;CR&gt; //断状态

\$ (addr)(H)(2)(0)&lt;CR&gt; //解除控制状态

应答: &gt;(addr)(2)(4)

注: 设置温度保存在 X5045 的 0X1BH, 0X1CH 单元中。

## (5) 比例参数 P 设定

格式: \$(addr)(P)(1)&lt;CR&gt; 第一路参数 P

\$ (addr)(P)(2)&lt;CR&gt; 第二路参数 P

应答: &gt;(addr)(5)

## (6) 周期参数 T 设定

格式: \$(addr)(T)(1)&lt;CR&gt; 第一路参数 T

\$ (addr)(T)(2)&lt;CR&gt; 第二路参数 T

应答: &gt;(addr)(6)

## 2. 全局变量及函数定义

#include &lt;at89S52.h&gt;

#define uchar unsigned char

#define uint unsigned int

int data real\_temp[2], period; /\* 全局变量定义 \*/

uchar time1, time2, period /\* 全局变量定义, 通道 1、2 的定时中断计



```

                                数,周期中断次数 */
uchar open[2],P[ch],r_char,r_byte,r_buffer[14];
sbit relay_1 = P2^2;
sbit relay_2 = P2^3;
uchar bdata flag;
sbit comm_flag = flag^0;          /* 通信标志 */
sbit self_flag = flag^1;          /* 自检标志 */
sbit ch1_ds18 = flag^2;
sbit ch2_ds18 = flag^2;          /* 通道 1、2 的 DS18B20 标志 */
sbit r_start = flag^7;            /* 接收开始信号标志 */
sbit r_end = flag^6;              /* 接收结束信号标志 */
sbit DE = P3^7;                   /* MAX485 发送、接收控制 */
sbit Led1 = P2^4;                  /* 通道 1 状态灯 */
sbit Led2 = P2^5;                  /* 通道 2 状态灯 */
uchar code period = 250;          /* 控制周期 */

```

### 3. 主程序清单

```

void main (void)
{
    comm_flag = 0;                /* 清标志位 */
    initialize();                  /* 初始化子程序 */
    while (self_flag == self_check();) /* self_flag = 0: 自检正常;
                                        self_flag = 1: 自检错误 */
    {
        err_dispose(0);           /* 错误处理子程序 */
        display(0);                /* 调显示子程序 */
        while(1)
        {
            dog_feed();            /* 喂狗子程序 */
            if(ch1_ds18 == read_temp(1))
            {
                err_dispose(1);
                display(1);         /* 通道 1 温度显示 */
                control(1);         /* 通道 1 温度控制 */
            }
            if(ch2_ds18 == read_temp(2))
            {
                err_dispose(2);
                display(2);         /* 通道 2 温度显示 */
                control(2);         /* 通道 2 温度控制 */
            }
            if(comm_flag)
            {
                {comm_dispose();} /* 通信处理模块 */
            }
        }
    }
}

```

#### 4. 继电器控制模块

因为固态继电器的输入控制属于两位控制,只有开或关两个状态,不能进行线性控制,所以其控制算法采用 PWM 控制,通过改变脉冲宽度(即 PWM 信号的占空比)的方法控制加热棒的加热时间。计算方法如下:首先设定控制周期  $T$ 、比例系数  $P$ ,它们可以通过计算机进行调整,继电器的导通、截止时间通过定时器控制。

本系统控制中,定时器每次中断定时 10 ms,若周期  $T$  为 2 500 ms,则需要中断 250 次,那么控制时所能实现的最小占空比的变化量为  $10/2\ 500=1/250$ ,占空比的范围为 0~1。实际计算时一个周期内( $N$  次中断,本例为 250 次中断,此周期可变)高电平输出的次数为  $M$ ,低电平输出的次数则为  $N-M$ ,根据偏差  $e$  的大小以及比例系数计算  $M=e\times P$ 。有时为了及时控制,也可以分段控制。如当偏差  $e_1<e<e_2$  时,采用 PWM 控制, $e<e_1$  时,固态继电器始终导通, $e>e_2$  时,固态继电器保持截止,具体  $e_1$ 、 $e_2$  的数值根据情况确定。

```
/* 控制程序,入口参数:通道号,函数返回:无 */
void control(uchar ch)
{
    int t;
    t = real_temp[ch] - set_temp[ch]; /* 计算实时测量温度与设置温度的偏差 */
    open[ch] = t * P[ch];
}

/* 定时中断函数 */
void timer0 () interrupt 1 using 2 /* 使用寄存器 2 区 */
{
    TH0 = - 9216 / 256;
    TL0 = - 9216 % 256; /* 赋初值,定时:10ms */
    if ( ++time1 <= open[0] )
    {
        Led1 = 0; /* 通道状态灯,灯亮 */
        relay_1 = 0; /* 继电器状态,导通 */
    }
    else { if (time1 >= period) /* 控制周期到,清 timer1 */
        time1 = 0;
        relay_1 = 1; Led1 = 1; /* 继电器截止,状态灯灭 */
    }

    if ( ++time2 <= open[1] ) { /* 通道 2 控制,同通道 1 */
        Led2 = 0;
        relay_2 = 0;
    }
    else
    { if (time2 >= period)
        time2 = 0;
        { relay_2 = 1; Led2 = 1; }
    }
}
```

## 5. 串口发送、接收模块,包括发送函数、接收函数,接收采用中断形式

```

/* 接收中断程序 */
serial() interrupt 4 using 1      /* 使用寄存器 1 区 */
{
    ES = 0;                       /* 关串口中断 */
    if (RI)
    {
        r_char = SBUF;            /* r_char 为全局变量, unsigned char */
        RI = 0;                  /* 清标志 */
        switch(r_char)
        {
            case 0x24:
                {
                    r_start = 1;    /* $ */
                    r_byte = 0;
                    break;
                }
            case 0x40: {r_start = 1; /* @ */
                        r_byte = 0;
                        break;}
            case 0x0D: {r_end = 1;   /* CR, 回车符 */
                        if (r_start)
                        {
                            receive_byte(); /* 接收处理程序 */
                            r_end = 0;
                            r_start = 0;    /* 清开始、结束标志 */
                            clear_buffer(); /* 清接收缓冲区 */
                        }
                        break;
                    }
            default: break;
        }
        if (r_byte >= 11) /* 接收字符个数到, 置位结束标志 */
            r_end = 1;
        if(r_start)
            /* 接收开始标志置位后, 依次将接收字符存接收缓冲区 r_buffer 数组 */
            {
                r_buffer[r_byte] = r_char;
                if(r_end != 1)
                    r_byte++;
            }
    }
}

```



```
    }  
    }  
    else if (TI)  
    TI = 0;  
    ES = 1;          /* 开串口中断 */  
    }  
    /* 发送程序,入口参数:发送字符 chr */  
    void putch(unsigned char chr)  
    {  
        DE = 1;          /* 控制 MAX485 在发送状态 */  
        SBUF = chr;      /* 发送字符 */  
        TI = 0;  
        while(! TI);    /* 等待 */  
        TI = 0;  
        DE = 0;          /* 控制 MAX485 接收状态 */  
    }
```

## 11.3 基于 GSM/CDMA 的防盗报警系统

GSM/CDMA 技术应用于汽车防盗定位系统是我国移动通信网络迅速发展的充分体现,本防盗报警系统基于 GSM/CDMA 模块,采用组合式的构造方法。传统的汽车防盗系统报警范围仅为 100~200 m,当车主离开报警范围就无法接收报警信号,而且汽车一旦失窃无法及时采取应对措施,这给犯罪分子提供了可乘之机。基于 GSM/CDMA 技术的汽车防盗定位系统,可以将汽车的报警信号以拨打报警电话和发送 SMS 短信的方式传输到任何 GSM/CDMA 网络可以覆盖的地方。

### 11.3.1 系统的目标任务

系统的目标任务包括:

- 主控器对 GSM/CDMA 模块具有实时控制性;
- 确保 GSM/CDMA 工作的稳定性;
- 要求系统的抗干扰性和报警功能齐全;
- 用户可以对报警器进行实时管理;
- 报警系统具有灵活性,降低产品成本。

### 11.3.2 系统的总体设计

根据以上目标任务,为了使系统具有灵活性,这里采用分体组合的方法,使得外部的传感器和报警装置独立起来,整个系统的优点如下:

- 主控器能及时地对 GSM/CDMA 模块收到的短信息采取相应的处理,保证了系统的抗干扰性;

- 系统采用轮询握手的方式保证模块能够正常工作,避免系统进入死锁状态。
- 用户可以直接使用自己的手机对系统进行控制,如设防、撤防、修改密码、查询状态等。
- 系统具有远程性,不受距离的影响。
- 多路传感器和继电器输出保证了系统报警检测方位的齐全和遇警处理的强大功能。
- 加设匪警按钮,遇匪可以直接向家人或者 110 报警,确保车主的安全。

### 11.3.3 系统的结构框图及工作原理

根据系统的目标任务以及总体设计方案,本系统主要框图如图 11.3.1 所示,系统由主控机、传感器、继电器组、蜂鸣器、手机模块构成。各个部分主要功能如下。

(1) 控制主机,主控机核心为 AT89S52 单片机,主控机负责接收传感器的报警信号,向继电器和蜂鸣器发出动作控制命令。主控机和 GSM/CDMA Modem 之间通过 USB 或者 RS232 串口进行双向通信,接收来自 GSM/CDMA Modem 的控制信息,根据车主的控制命令进行系统设置,当接到传感器的报警信号后主控机要控制 GSM/CDMA Modem 拨打设定的电话号码或发送短信。

(2) GSM/CDMA Modem(手机),主要负责向主控机传送控制信息、向主控机指定的电话号码发送报警信息。

(3) 继电器、蜂鸣器,继电器在主控机的控制下执行锁死车门,切断电路、油路等保护动作,蜂鸣器发出报警声音。

(4) 报警传感器,可以选配红外报警探测器、振动传感器等,本系统最多允许接入 4 路报警传感器。

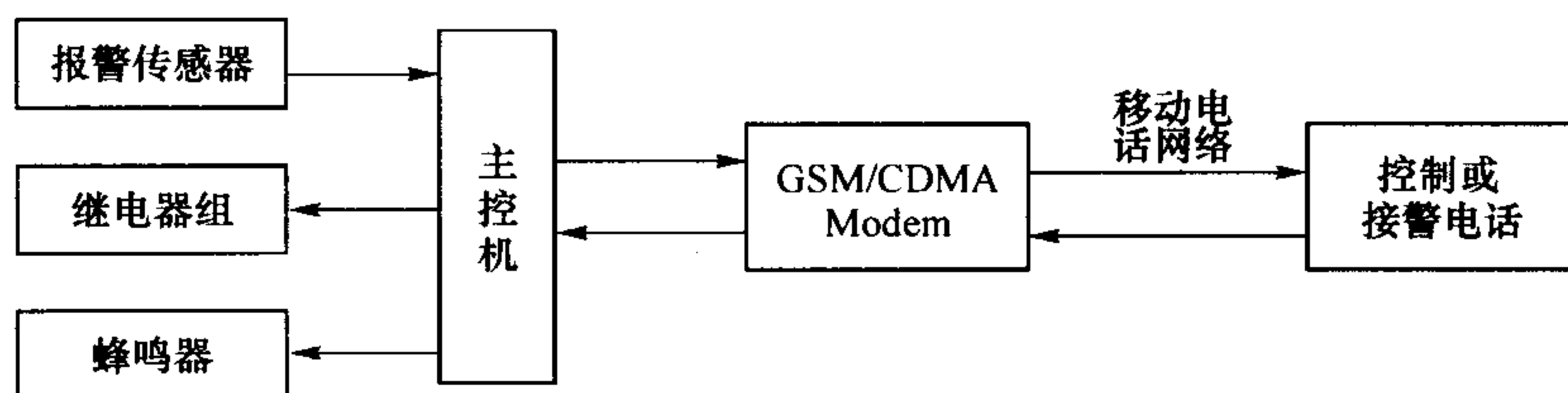


图 11.3.1 系统结构图

部分硬件原理图如图 11.3.2 所示。

### 11.3.4 程序流程及软件设计

本系统软件部分则基于现在市场上大多数 GSM/CDMA Modem 均支持 GSM07.05 规定的 AT 指令集。该指令集是 ETSI(欧洲通信技术委员会)发布的,其中包含了对拨号和收发 SMS 的控制。利用 GSM/CDMA 手机的串行接口,单片机向手机收发一系列的 AT 命令,就能达到控制手机拨号和收发 SMS 的目的。常用的 AT 指令如下:

ATD 拨号;	ATH 挂机;	ATA 接电话;
ATDL 重拨上一次电话号码;		AT+CSMS 选择短信息服务;
AT+CPMS 选择短信息内存;		AT+CMGF 选择短信息格式;
AT+CSCA 短信息中心地址;		AT+CNMI 显示新收到的短信息;

AT+CMGR 读短信息；

AT+CMGS 发送短信息；

AT+CMGL 列出 SIM 卡中短信息。

程序流程图如图 11.3.3 所示。

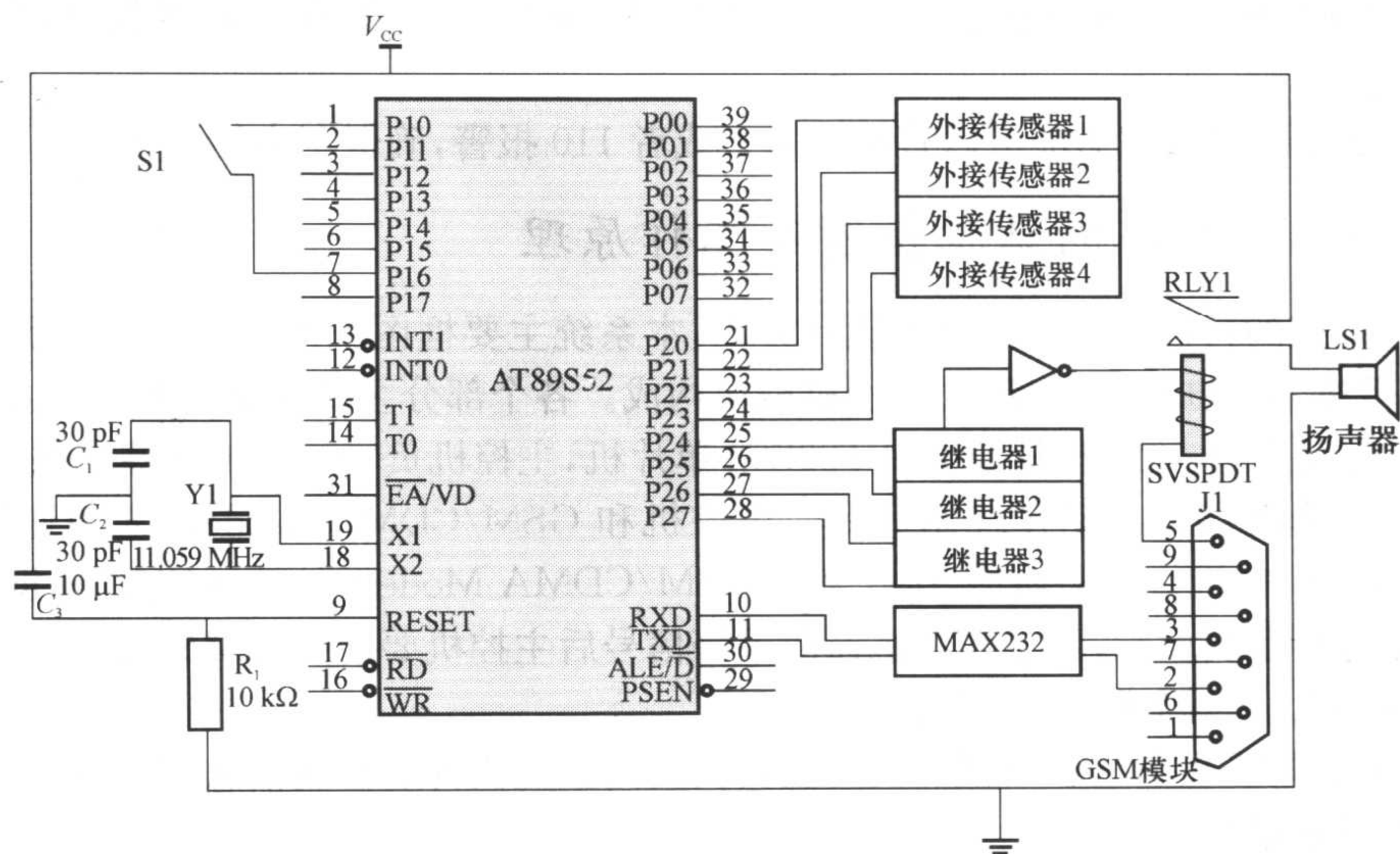


图 11.3.2 系统原理图(部分)

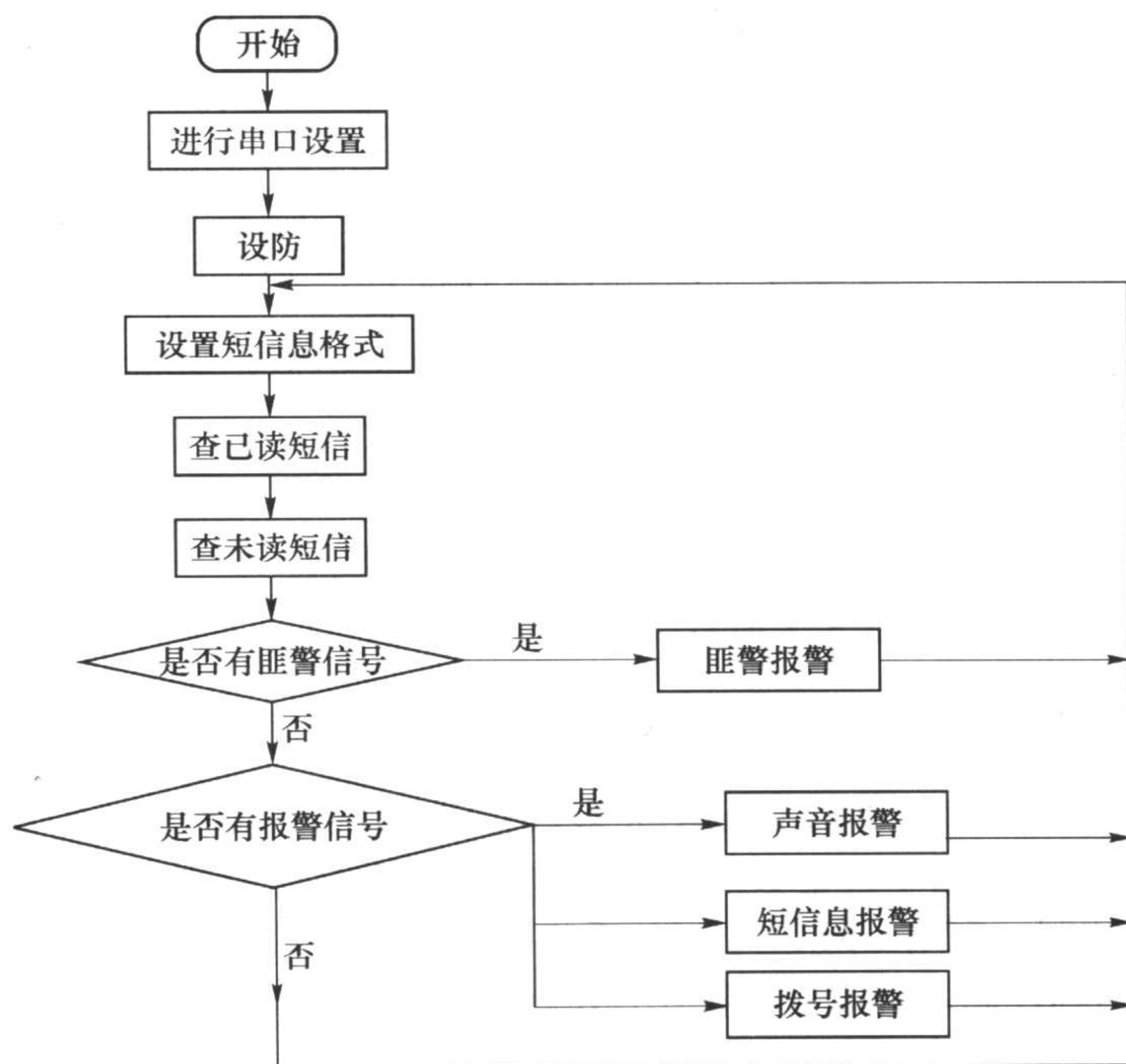


图 11.3.3 系统程序流程图

系统程序采用汇编语言编程,主程序共调用如下 5 个子程序。

- 设置短信息格式子程序:用来进行单片机 GSM/CDMA 模块的握手过程和设置 GSM/CDMA 模块发送短信息的格式。
- 查已读短信子程序:一般用于系统重新启动时,设置系统密码。
- 查未读短信子程序:系统工作期间,用户通过短信管理系统,主要功能是先检测密码,密码正确后,判断用户的指令,进行修改密码、撤防、设防、查询状态等功能。
- 发送匪警短信息子程序:执行发送匪警短信息功能。
- 报警子程序:执行控制报警喇叭、拨号、发送短信息报警。

### 1. 系统主程序

```

ORG 0000H
SJMP START
ORG 0050H
START: NOP
CLEAR:  MOV    @R0,A        ;系统缓冲一小段时间
        DJNZ    R0,CLEAR
        MOV     89H,#20H     ;波特率发生器 T1 工作在模式 2 上
        MOV     98H,#50H     ;串行口控制,工作方式 1 + 允许接收
        MOV     8DH,#253     ;定时器 1 中高 8 位放计数初值或 0FDH
        MOV     8BH,#253     ;定时器 1 中低 8 位用于计数(可不写)
        SETB    0AFH         ;中断总允许 EA
        SETB    0ACH         ;串行口中断允许 ES
        MOV     88H,#00H     ;包括 TR1,此处为了加强可靠性 88H,TCON
        SETB    8EH         ;定时器 1 开始工作

        MOV     R6,#11H      ;设防,11 为设防标记
MAIN:    NOP
        SETB    0A0H
        SETB    0A1H
        SETB    0A2H
        SETB    0A3H        ;传感器输入端置 1
        NOP
        CLR     0A4H         ;继电器输出端清零
        CLR     97H         ;匪警信号总线清零
        SETB    90H
WRIT:    JB      90H,BB      ;判断 S1 是否按下,如果没有按下就等待
        ACALL   DELAY10     ;延时 10 ms 消触点抖动
        JB      90H,WRIT     ;去除干扰信号
        JNB     90H,$        ;等待按键松开

```

```

ACALL  FAS1          ;发送匪警短信息

NOP

ACALL  DELAY10
ACALL  DELAY10
ACALL  DELAY10
NOP
NOP
BB:    ACALL  SZ          ;调用设置短信息格式
NOP
NOP
ACALL  JS          ;查已读短信
NOP
NOP
NOP
ACALL  DELAY10
NOP
NOP
ACALL  JS1          ;查未读短信
NOP
NOP
NOP
ACALL  DELAY10
ACALL  DELAY10
ACALL  DELAY10
ACALL  DELAY10
NOP
NOP
NOP
CJNE   R6, #22H, CHAX ;判断是否撤防, 22 是撤防标记
AJMP   MAIN
CHAX:
JNB    0A0H, JJJ
JNB    0A1H, JJJ
JNB    0A2H, JJJ
JNB    0A3H, JJJ      ;传感器有报警信号, 调用报警程序, 采取报警动作

```

```

NOP
AJMP  MAIN
JJJ:  ACALL  BAOJ      ;调用报警子程序
      AJMP  MAIN
      END

```

## 2. 报警子程序

```

BAOJ:  SETB  0A4H      ;发出报警信号
      NOP
      ACALL  FAS        ;发送报警短信
      NOP
      NOP
      ACALL  DELAY10
      ACALL  DELAY10
      ACALL  DELAY10
      NOP
      ACALL  BOH        ;拨号报警
      NOP
      NOP
      NOP
      ACALL  DELAY10
      ACALL  DELAY10
      NOP
      NOP
      SJMP  $
      RET

```

## 3. 设置短信息格式子程序

```

SZ:    CLR   99H        ;发送中断 TI 清零
      MOV   A, #41H      ;A 的 ASCII 码
      MOV   99H, A       ;99H 是缓冲区,把 A 的内容送缓冲区
      JNB   99H, $       ;判断 TI 是否为 1,不是则等待
      CLR   99H
      MOV   A, #54H      ;T 的 ASCII 码
      MOV   99H, A
      JNB   99H, $       ;判断 TI 是否为 1,不是则等待
      CLR   99H
      MOV   A, #0DH      ;回车的 ASCII 码
      MOV   99H, A
      JNB   99H, $       ;发送 AT,回车结束

```

```

NOP
NOP
CLR    0AFH          ;禁止总中断
KK1:   CLR    98H      ;接收中断 RI 清零
MOV    A,99H         ;99H 是缓冲区,把缓冲区的内容送 A
JNB    98H,$          ;判断 RI 是否为 1,不是则等待
CJNE   A,#4BH,KK1     ;接收到返回 OK 为结束
NOP
NOP
BBB:   SETB   0AFH     ;允许总中断
MOV    DPTR,#GSM       ;发送 AT + CMGF = 0
GM10:  CLR    99H      ;发送中断 TI 清零
MOV    A,#00H
MOVC   A,@A+DPTR
MOV    99H,A
JNB    99H,$
CJNE   A,#0DH,GM20     ;判断是否是最后一个字节,不是则继续
SJMP   GM30
GM20:  INC    DPTR
ACALL  DELAY10
SJMP   GM10
GM30:  NOP
NOP
NOP
NOP
NOP
GSM:   DB 41H,54H,2BH,43H,4DH,47H,46H,3DH,30H,0DH      ;AT + CMGF = 0
CLR    0AFH
KK:    CLR    98H      ;接收中断 RI 清零
MOV    A,99H         ;把缓冲区的内容送 A
JNB    98H,$          ;如果 RI 为 0 说明发送没有完成,程序等待 T
CJNE   A,#4BH,KK      ;接收到返回 OK 为结束
NOP
NOP
NOP
NOP
RET

```

## 4. 拨号报警子程序

```

BOH:   SETB  0AEH           ;允许总中断
        NOP
        CLR   99H           ;发送中断 TI 置零
        MOV   A, #41H       ;41H 是 A 的 ASCII 码
        MOV   99H, A        ;把发送字符送缓冲区
        JNB   99H, $        ;如果 TI 为 0 说明发送没有完成,程序等待
        CLR   99H
        MOV   A, #54H       ;54H 是 T 的 ASCII 码
        MOV   99H, A        ;把发送字符送缓冲区
        JNB   99H, $        ;如果 TI 为 0 说明发送没有完成,程序等待
        CLR   99H          ;发送中断置零
        MOV   A, #44H       ;44H 是 D 的 ASCII 码
        MOV   99H, A        ;把发送字符送缓冲区
        JNB   99H, $        ;如果 TI 为 0 说明发送没有完成,程序等待
        MOV   R0, #50H      ;把电话号码的首地址送 R0
CM:     CLR   99H
        MOV   A, @R0
        MOV   99H, A
        JNB   99H, $        ;如果 TI 为 0 说明发送没有完成,程序等待
        INC   R0
        CJNE  R0, #5AH, CM   ;判断电话号码是否发送完毕,没有就继续
        RET

```

本章通过两个实例具体介绍了单片机应用系统的开发过程,实际开发过程必须是“软硬兼施”,硬件设计与软件设计综合考虑、顾全大局,才能设计出高性能的单片机应用系统。

## 习 题

1. 说明单片机系统设计需要具备的知识和能力,以及开发的步骤。
2. 查找 DS18B20 的 DATASHEET,按照其说明编写 DS18B20 的相关程序,程序编写用两种语言:汇编语言和 C 语言。
3. 查找 E2PROM 24C02 的相关资料,绘制其与单片机 AT89S52 之间的电路连接图,并编写相应的读写程序。
4. 查找有关 MAX485、MAX1487、MAX1480 的资料,设计其与单片机的连接电路,并编写相应的程序。
5. 查找 GSM 模块资料,了解 GSM 模块的使用方法,熟悉 AT 指令集。



## 实 践 训 练

学完本章内容后,读者已经掌握了 AT89S52 单片机的硬件原理、编程调试、系统扩展、总体设计的方法,可按照下述步骤进行整体训练。

1. 在第 10 章实践训练 2 的基础上,连接 DS18B20,利用数码管显示其测量温度值。
2. 按照图 11.2.3 所示电路搭建硬件电路,利用 C 语言编写数码管显示程序和总体调试程序。
3. 设计 8 路热电偶温度控制系统,由 DS18B20 做冷端温度补偿,利用数码管显示实测温度,键盘输入控制的温度值,交流固态继电器控制 220 V 加热丝。设计硬件电路、程序流程图并编写相应的程序。

## 第 12 章 其他系列单片机介绍

目前应用中的单片机品种繁多,本章介绍两种在我国广泛应用的单片机,然后再介绍一些著名的有代表性的单片机及其生产厂商。

### 12.1 HOLTEK 公司的 HT48×× 系列单片机概述

台湾 HOLTEK 公司的一系列单片机属后起之秀,其特点是性价比高,在市场中占有一定的份额,并受到越来越多的客户所喜爱,以快速的发展势头冲击着几乎是国外半导体商一统天下的单片机市场,其产品已广泛应用于电话机、电扇及灯光控制器、电子秤量器具、洗衣机程控器等家用电器产品。HOLTEK 公司的单片机以型号多、品种全、应用范围广等特点著称。

#### 12.1.1 HT48×× 系列单片机的主要性能

HT48×× 系列单片机属于多 I/O 接口精简指令集高性能单片机,为适应不同产品的设计要求,这个系列有简易经济型以及要求较高、内部资源相对丰富的高级型,它们的主要区别在于内部的 I/O 接口数目不等, RAM 和 ROM 容量不同,定时/计数器的长度有别,此外它们的堆栈层数也不尽一致。从表 12.1.1 中即可看出它们的主要区别。

HT48 系列单片机有以下几种型号: HT48C10-1、HT48C50-1、HT48CA0、HT48R10-1、HT48R30A-1、HT48R50A-1、HT48R05A-1、HT48R06A-1 和 HT48RA0A 型。其中, C 型单片机是指掩膜型,而 R 型单片机则指 OTP 型,这是 C 型和 R 型单片机的区别。

表 12.1.1 HOLTEK 系列 HT48×× 单片机主要特性列表

型号 配置	48C10-1	48C50-1	48CA0	48R10A-1	48R30A-1	48R50A-1	48R05A-1	48R06A-1	48RA0A
I/O 接口	21	35	10	21	25	35	13	13	10
RAM	64×8	160×8	32×8	64×8	96×8	160×8	32×8	64×8	32×8
ROM	1 024×14	4 096×15	1 024×14	1 024×14	2 048×14	4 096×15	512×14	1 024×14	1 024×14
内中断	1	2	0	1	1	2	1	1	none
外中断	1	1	0	1	1	1	1	1	none
定时器	1(8)	2(8,16)	none	1(8)	1(8)	2(8,16)	1(8)	none	—
堆栈	4	6	1	4	4	6	2	2	1
WDT	—	—	—	—	—	—	—	—	—

HT48R05A-1 是 8 位高性能单片机,它以价廉物美而著称,专为多 I/O 接口的产品而设计的 OTP 型号,可方便地应用于简单的控制系统。

### 12.1.2 HT48××系列单片机的引脚描述

基本 I/O 型 HT48R05A-1 的引脚图如图 12.1.1 所示,其引脚描述如下。

- Pin1~Pin4、Pin15~Pin18: PA0~PA7, 双向 8 位输入/输出端口, 掩膜选项可选择上拉电阻, 每一位均可被设置为唤醒输入, 软件指令确定 CMOS 输出。或带上拉电阻的施密特触发输入(由上拉电阻选项确定)。
- Pin5、Pin6、Pin7: PB2、PB1/BZ、PB0/ $\overline{\text{BZ}}$ , 双向 3 位输入/输出端口, 每一位均可由掩膜选项设置为唤醒输入, 软件指令确定 CMOS 输出。或带上拉电阻的施密特触发输入(由上拉电阻选项确定)。PB0 和 PB1 与 BZ 和  $\overline{\text{BZ}}$  共享一个引脚, 一旦 PB0 和 PB1 被选为蜂鸣器输出, 输出信号来自内部的 PFD 发生器(与定时/计数器共享)。
- Pin12、Pin8:  $V_{\text{DD}}$ ,  $V_{\text{SS}}$ , 正、负电源端。
- Pin9、Pin10: PC0/INT、PC1/TMR。双向 I/O 接口, 软件指令确定 CMOS 输出, 或带上拉电阻的施密特触发输入(由上拉电阻选项确定)。外部中断和定时器输入与 PC0 和 PC1 共享一个引脚, 外部中断输入是电平由高向低变化的激励信号。
- Pin11:  $\overline{\text{RES}}$ 。施密特触发复位端, 低电平有效。
- Pin13、Pin14: OSC1、OSC2。石英晶振或 RC 振荡。OSC1 和 OSC2 被连接到一个 RC 或一个石英晶体(由掩膜选项确定)来产生内部系统时钟, 在 RC 方式下 OSC2 是一个系统时钟 4 分频输出端。

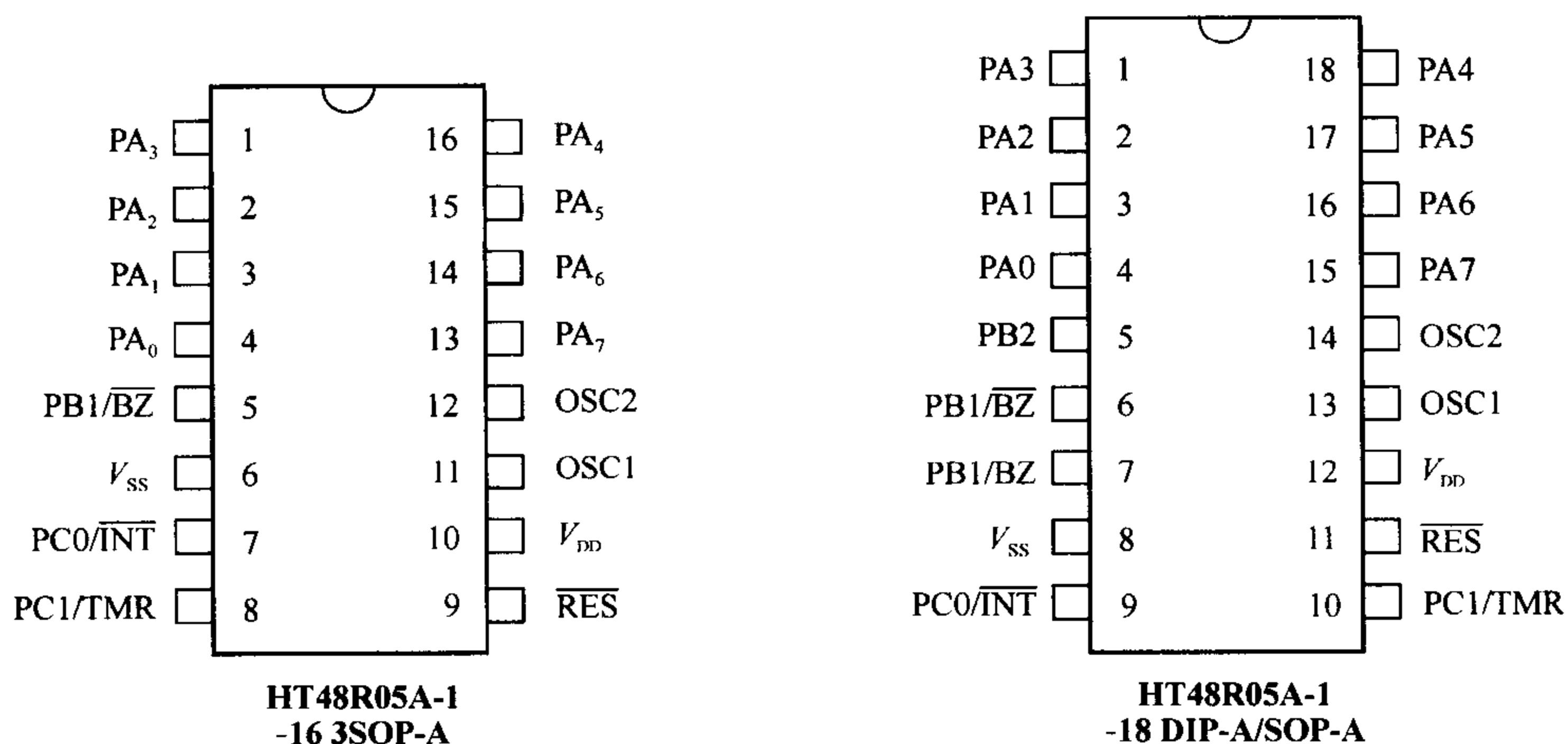


图 12.1.1 HT48R05A-1 引脚图

### 12.1.3 HT48××系列单片机的内部结构框图

图 12.1.2 所示是 HT48R05A-1 内部结构框图。

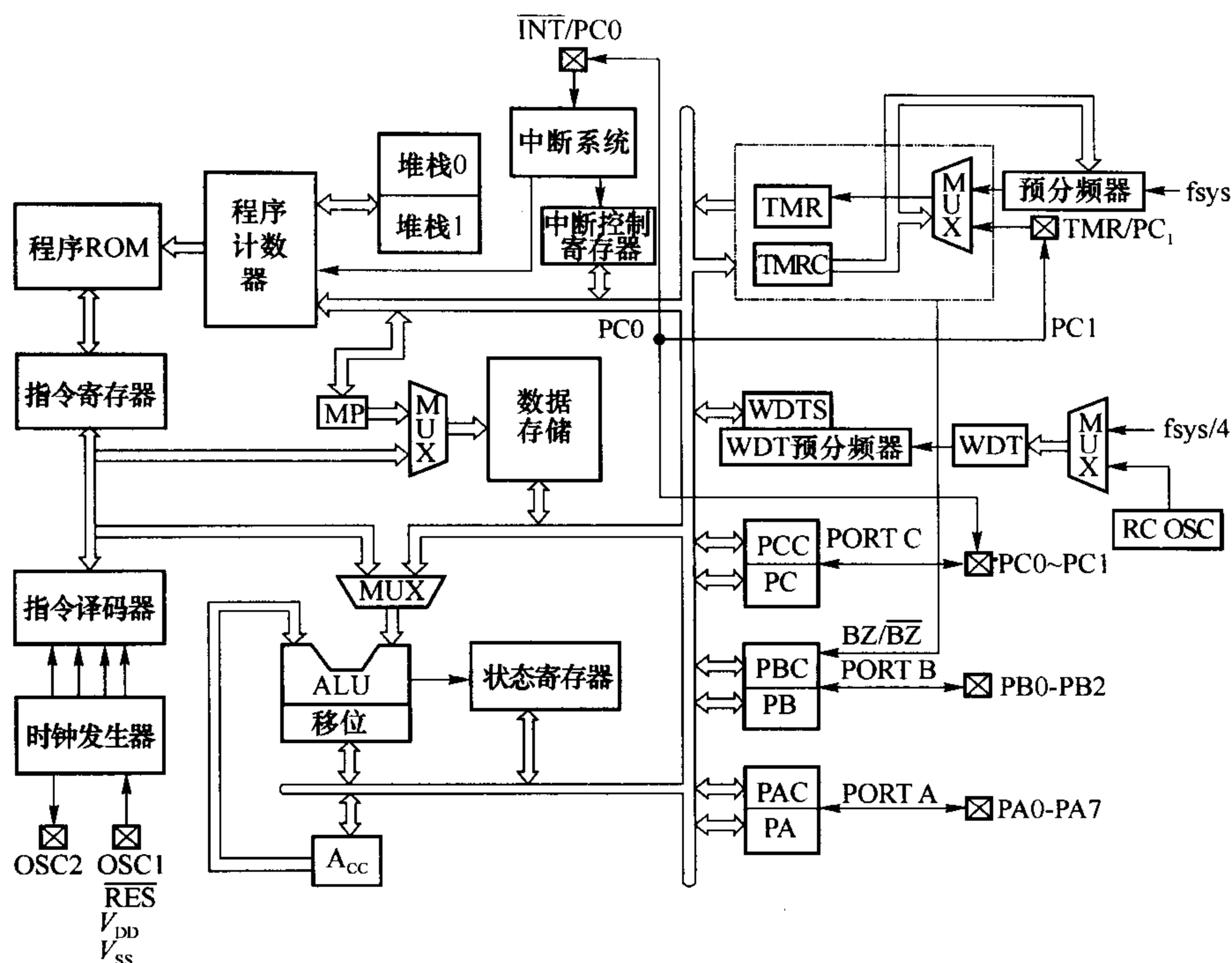


图 12.1.2 HT48R05A-1 内部结构框图

### 12.1.4 HT48××系列单片机的指令集

由于资源大小问题,HT48CA0/HT48RA0A 和 HT48R05A-1 单片机只有 62 条指令,其余均为 63 条指令,HT48CA0/HT48RA0A 无 RETI 指令,HT48R05A-1 无 TABRDL 指令。

指令寻址方式有下面 5 种。

(1) 立即寻址:此寻址法是将立即数紧跟在运算码(opcode)后,例如,

MOV A,33H ;ADDA,33H

(2) 直接寻址:直接寻址的情况只允许在存储器之间进行数据传送,例如,

MOV [33H],A ;CLR[33H]

(3) 间接寻址:在间接寻址方式中必定要使用到间接寻址暂存器(R0,R1),例如,

MOV R0,A ;MOV A,R0

(4) 特殊暂存器寻址:此寻址方式完全是针对某一暂存器作运算,例如,

CLR WDT ;CLR WDT1

(5) 指针寻址:指针寻址只适用在配合作查表指令,例如,

MOV A,02H ;MOV TBLP,A

表 12.1.2~表 12.1.4 是 HT48 系列单片机指令集摘要。

表 12.1.2 HT48 系列单片机指令集摘要 1

助记符	说 明	指令周期	影响标志位
算术运算			
ADD A,[m]	累加器与数据存储器相加结果放入累加器	1	Z,C,AC,OV
ADDM A,[m]	累加器与数据存储器相加结果放入数据存储器	11	Z,C,AC,OV
ADD A,x	累加器与立即数相加结果放入累加器	1	Z,C,AC,OV
ADC A,[m]	累加器与数据存储器进位标志相加结果放入累加器	1	Z,C,AC,OV
ADCM A,[m]	累加器与数据存储器进位标志相加结果放入数据存储器	11	Z,C,AC,OV
SUB A,x	累加器与立即数相减结果放入累加器	1	Z,C,AC,OV
SUB A,[m]	累加器与数据存储器相减结果放入累加器	1	Z,C,AC,OV
SUBM A,[m]	累加器与数据存储器相减结果放入数据存储器	11	Z,C,AC,OV
SBC A,[m]	累加器与数据存储器进位标志相减结果放入累加器	1	Z,C,AC,OV
SBCM A,[m]	累加器与数据存储器进位标志相减结果放入数据存储器	11	Z,C,AC,OV
DAA [m]	将加法运算后放入累加器的值调整为十进制数并将结果放入数据存储器	11	C
逻辑运算			
AND A,[m]	累加器与数据存储器做与运算结果放入累加器	1	Z
OR A,[m]	累加器与数据存储器做或运算结果放入累加器	1	Z
XOR A,[m]	累加器与数据存储器做异或运算结果放入累加器	1	Z
ANDM A,[m]	累加器与数据存储器做与运算结果放入数据存储器	11	Z
ORM A,[m]	累加器与数据存储器做或运算结果放入数据存储器	11	Z
XORM A,[m]	累加器与数据存储器做异或运算结果放入数据存储器	11	Z
AND A,x	累加器与立即数做与运算结果放入累加器	1	Z
OR A,x	累加器与立即数做或运算结果放入累加器	1	Z
XOR A,x	累加器与立即数做异或运算结果放入累加器	1	Z
CPL [m]	对数据存储器取反结果放入数据存储器	11	Z
CPLA [m]	对数据存储器取反结果放入累加器	1	Z
递增和递减			
INCA [m]	数据存储器的内容加 1 结果放入累加器	1	Z
INC [m]	数据存储器的内容加 1 结果放入数据存储器	11	Z
DECA [m]	数据存储器的内容减 1 结果放入累加器	1	Z
DEC [m]	数据存储器的内容减 1 结果放入数据存储器	11	Z

表 12.1.3 HT48 系列单片机指令集摘要 2

助记符	说 明	指令周期	影响标志位
移 位			
RRA[m]	数据存储器右移一位结果放入累加器	1	无
RR[m]	数据存储器右移一位结果放入数据存储器	1 <sup>①</sup>	无
RRCA[m]	带进位将数据存储器右移一位结果放入累加器	1	C
RRC[m]	带进位将数据存储器右移一位结果放入数据存储器	1 <sup>①</sup>	C
RLA[m]	数据存储器左移一位结果放入累加器	1	无
RL[m]	数据存储器左移一位结果放入数据存储器	1 <sup>①</sup>	无
RLCA[m]	带进位将数据存储器左移一位结果放入累加器	1	C
RLC[m]	带进位将数据存储器左移一位结果放入数据存储器	1 <sup>①</sup>	C
数据传送			
MOV A,[m]	将数据存储器送至累加器	1	无
MOV[m],A	将累加器送至数据存储器	1 <sup>①</sup>	无
MOV A,x	将立即数送至累加器	1	无
位运算			
CLR[m].i	将数据存储器的第 i 位清 0	1 <sup>①</sup>	无
SET[m].i	将数据存储器的第 i 位置 1	1 <sup>①</sup>	无
转 移			
JMP addr	无条件跳转	2	无
SZ [m]	如果数据存储器为 0 则跳过下一条指令	1 <sup>②</sup>	无
SZA [m]	数据存储器送至累加器如果内容为 0 则跳过下一条指令	1 <sup>②</sup>	无
SZ[m].i	如果数据存储器的第 i 位为 0 则跳过下一条指令	1 <sup>②</sup>	无
SNZ[m].i	如果数据存储器的第 i 位不为 0 则跳过下一条指令	1 <sup>②</sup>	无
SIZ[m]	数据存储器加 1 如果结果为 0 则跳过下一条指令	1 <sup>③</sup>	无
SDZ[m]	数据存储器减 1 如果结果为 0 则跳过下一条指令	1 <sup>③</sup>	无
SIZA[m]	数据存储器加 1 将结果放入累加器如果结果为 0 则跳过下一条指令	1 <sup>②</sup>	无
SDZA[m]	数据存储器减 1 将结果放入累加器如果结果为 0 则跳过下一条指令	1 <sup>②</sup>	无
CALL addr	子程序调用	2	无
RET	从子程序返回	2	无
RET A,x	从子程序返回并将立即数放入累加器	2	无
RETI	从中断返回	2	无

表 12.1.4 HT48 系列单片机指令集摘要 3

助记符	说 明	指令周期	影响标志位
查 表			
TABRDC[m]	读取当前页的 ROM 内容并送至数据存储器 and TBLH	2	无
TABRDL[m]	读取最后页的 ROM 内容并送至数据存储器 and TBLH	2 <sup>①</sup>	无
其他指令			
NOP	空指令	1	无
CLR[m]	清除数据存储器	1 <sup>①</sup>	无
SET[m]	置位数据存储器	1 <sup>①</sup>	无
CLR WDT	清除看门狗定时器	1	TO, PD
CLR WDT1	预清除看门狗定时器	1	TO <sup>④</sup> , PD <sup>④</sup>
CLR WDT2	预清除看门狗定时器	1	TO <sup>④</sup> , PD <sup>④</sup>
SWAP [m]	交换数据存储器的高低字节结果放入数据存储器	1 <sup>①</sup>	无
SWAPA [m]	交换数据存储器的高低字节结果放入累加器	1	无
HALT	进入暂停模式	1	TO, PD

注: x 立即数, m 数据存储器地址, A 累加器 ACC, i 第 0~7 位, addr 程序存储器地址, ①如果数据是加载到 PCL 寄存器则指令执行周期会被延长一个指令周期 4 个系统时钟, ②如果满足跳跃条件则指令执行周期会被延长一个指令周期 4 个系统时钟否则指令执行周期不会被延长, ③包含①和②, ④如果执行 CLW WDT1 或 CLR WDT2 指令后看门狗定时器被清除则会影响 TO 和 PD 标志位, 否则不会影响 TO 和 PD 标志位。

## 12.2 PIC16C5×系列单片机概述

PIC 系列单片机是美国 Microchip 公司推出的具有精简指令集的高性能 8 位单片机, 其优点是引脚少, 性能优越, 可直接带 LED 负载, 具有低功耗省电模式, 可广泛应用于复杂程度较低、功耗较低场合。十多年来 PIC 系列单片机正以迅猛的速度发展。

### 12.2.1 PIC16C5×系列单片机的主要性能

PIC16C5×属 CMOS 单片机, 是一个低价位高性能 8 位单片机, 使用了仅有 33 条单字节单周期指令的精简指令集, 每条指令执行时间最快可达 200 ns。易于记忆和使用的指令系统可大大减少产品的开发时间。多种时钟振荡电路、睡眠、低功耗、省电模式及 WDT(看门狗)代码保护功能, 这些特性具有较大优势。PIC16C5×系列单片机可广泛应用于电机控制、汽车电路、家用电器等领域。主要性能如下:

- RISC 精简指令集, 指令仅有 33 条, 指令长度为 12 位;
- 绝大部分均为单机器周期指令;
- 工作速度高, 最快可达 200 ns(20 MHz 时钟时)。数据长度为 8 位;
- 片内程序存储器容量为 512 B~2 KB;
- 片内静态数据存储器(SRAM)为 25~73 B;
- 硬件组成 7 个专用寄存器;
- 两级硬件堆栈;

- 有直接、间接、相对和位寻址功能；
- 12~20 条 I/O 引脚, 每条引脚均可设置为输入和输出状态；
- 多种时钟振荡电路及 WDT 定时器电路；
- 宽工作电压范围和低功耗模式, 工作电压为 2.5~6.0 V, 典型工作电流为 2 mA, 睡眠状态仅为 3  $\mu$ A；
- 程序保密位, 有效地保护用户的产权。

PIC16C5 $\times$ 系列单片机有多种不同的程序存储器和数据存储器、I/O 引脚、振荡类型、振荡频率及封装形式。这些器件可为研发实验提供方便, 4 种振荡方式是 RC、XT、HS 和 LP, 所需的振荡器是通过对片内的 EPROM 编程实现的, 未编程状态器件为默认的 RC 振荡方式。表 12.2.1 是 PIC16C5 $\times$ 系列单片机一览表。

表 12.2.1 PIC16C5 $\times$ 系列单片机一览表

型号	EPROM	RAM	时钟频率	定时器	工作电压/V	I/O 口数	封装形式
PIC16C52	384×12	25	0~4 MHz	1	2.5~6.0	12	18DIP/SOIC
PIC16C54	512×12		0~20 MHz	1+WDT		20	28DIP/SOIC
PIC16C55						12	18DIP/SOIC
PIC16C56	1 024					72	20
PIC16C57	2 048	73				12	18DIP/SOIC
PIC16C58							

## 12.2.2 PIC16C5 $\times$ 系列单片机的引脚描述

PIC16C5 $\times$ 系列单片机有两种封装形式, 一种是双列直插方式, 另一种是表面贴装方式。其引脚如图 12.2.1 所示。现对其功能引脚简述如下。

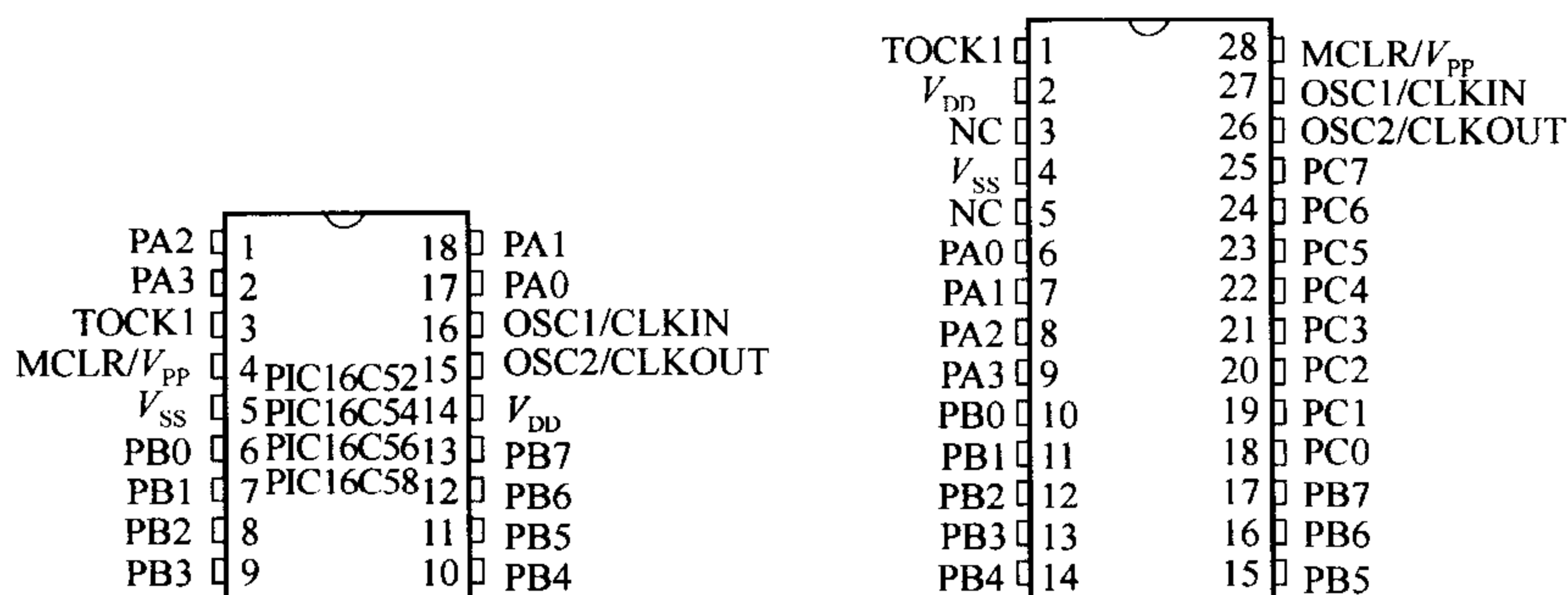


图 12.2.1 PIC16C5 $\times$ 单片机引脚图

- RA0~RA3: I/O 输入输出口 A, 对应寄存器 F5, 是一个 4 位 I/O 端口, 可位控。
- RB0~RB7: I/O 输入输出口 B, 对应寄存器 F6, 是一个 8 位 I/O 端口, 可位控。
- RC0~RC3: I/O 输入输出口 C, 对应寄存器 F7, 是一个 8 位 I/O 端口, 可位控。只



有 PIC16C55 和 PIC16C57 才有。

- RTCC: 实时时钟/计数器输入端, 在此端口输入信号的上升沿或下降沿计数, 边沿可通过软件选择。
- MCLR: 主复位端, 当 MCLR 为低电平时对单片机复位。
- OSC1: 振荡信号输入端。这个端口用于外部振荡信号的输入, 用 RC 振荡时, 它接 RC 电路, 用石英振荡电路时, 接石英晶体一端。
- OSC2: 振荡信号输出端。在用石英晶体振荡器或陶瓷振荡器时通过一个串联电阻  $R$  接振荡晶体一端, 在 RC 振荡时常作 CLKOUT 输出 ( $CLKOUT = \frac{1}{4} f_{osc}$ )。
- $V_{DD}$ : 电源电压。一般为 5 V, 其范围在 2.5~6.25 V 之间。
- $V_{SS}$ : 地端。
- NC: 空引脚。

### 12.2.3 PIC16C5×系列单片机的内部结构框图

图 12.2.2 是 PIC16C5×内部结构框图。

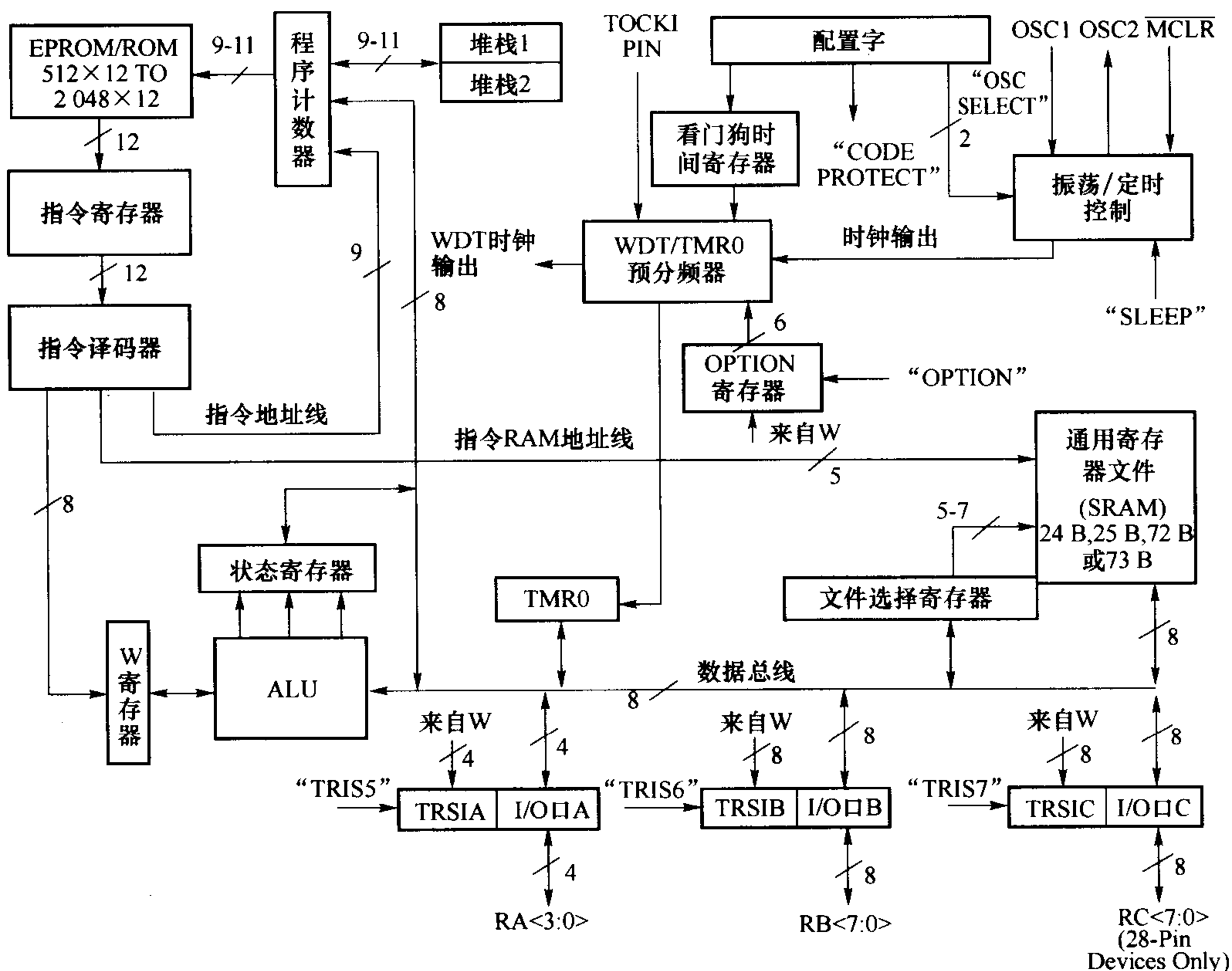


图 12.2.2 PIC16C5×内部结构框图

### 12.2.4 PIC16C5×系列单片机的指令集

PIC 系列单片机按其产品的规模,指令系统也不尽相同,分基本级产品、中级型产品和高级型产品 3 类。最基本的 PIC16C5×系列单片机每条指令的字长为 12 位,只有 33 条指令。中级产品的指令字长为 14 位,有 37 条指令。而高级产品的指令字长却为 16 位,共有 58 条指令。

根据 PIC16C5×系列单片机的指令系统,这 33 条指令可分成 3 类:面向字节操作类指令、面向位操作类指令、面向常数和控制操作类指令。对于字节操作类指令, $f$  是文件寄存器指针, $d$  为目标指针。文件寄存器用于确定 32 个文件寄存器中指令所用的寄存器。对于 PIC16C57 单片机,FSR(专用寄存器)中位 5 和 6 用于选择寄存器块。目标指针声明操作结果的存放地址,如果  $d$  为 0,声明结果在  $w$  寄存器,若  $d$  为 1,结果则送至指令指明的文件寄存器中。

对于位操作指令, $b$  是位指针,它用于标明操作的位, $f$  是文件寄存器指针,它用于指明位所在的文件寄存器。

PIC 系列单片机的一个指令周期需 4 个时钟周期。除了条件测试为真或程序计数器被指令执行所改变,例如 DECFSZ 或 CALL 指令外,所有指令执行仅需一个指令周期。假设时钟振荡频率选为 4.0 MHz,一个指令周期就是  $1\ \mu\text{s}$ ,那么绝大部分的指令运行只需  $1\ \mu\text{s}$ ,而上述少数的指令执行时间则需  $2\ \mu\text{s}$ 。

基本级产品指令系统实用于 PIC16C5×系列单片机,如 PIC16C52/54/56/57/58 等产品。而中级型产品指令系统则实用于 PIC16C6×/7×/8×等中级产品。高级型指令系统实用于 PIC17C××系列高级单片机。表 12.2.2 是 PIC16C5×系列单片机的指令集。

表 12.2.2 PIC16C5×系列单片机的指令集

助记符	说 明	影响标志位
字节操作指令		
ADDWF $f,d$	$w,f$ 相加	$w,DC,Z$
ANDWF $f,d$	$w,f$ 相“与”	$Z$
CLRF $f$	$f$ 清 0	$Z$
CLR $w$	$w$ 清 0	$Z$
COMF $f,d$	$f$ 求反	$Z$
DECF $f,d$	$f$ 减 1	$Z$
DECFSZ $f,d$	$f$ 减 1,为零则转移	NONE
INCF $f,d$	$f$ 加 1	$Z$
INCFSZ $f,d$	$f$ 加 1,为零则转移	NONE
IORWF $f,d$	$w,f$ 或运算	$Z$
MOVF $f,d$	传送 $f$	$Z$
MOVWF $f$	传送 $w$ 至 $f$	NONE
NOP	控操作	NONE
RLF $f,d$	循环左移	$C$
RRF $f,d$	循环右移	$C$
SUBWF $f,d$	从 $f$ 中减去 $w$	$C,DC,Z$
SWAPF	$f$ 高低半字节互换	NONE
XORWF $f,d$	$w,f$ 异或	$Z$

续表

助记符	说 明	影响标志位
位操作指令		
BCF f,b	清 f 的位 b	NONE
BSF f,b	置 f 的位 b	NONE
BTFSC f,b	f 的位 b 为 0 则转移	NONE
BTFSS f,b	f 的位 b 为 1 则转移	NONE
立即数和控制操作指令		
ANDLW k	立即数和 w 相与	Z
CALL k	调用子程序	NONE
CLRWDT k	清看门狗定时器	TO,PD
GOTO k	转移至指定地址	NONE
IORLW k	立即数和 w 相或	Z
MOVLW k	传送立即数至 w	NONE
OPTION	装 OPTION 寄存器	NONE
RETLW k	w 带参数返回	NONE
SLEEP	进入睡眠方式	TO,PD
TRIS f	设置 I/O 口状态	NONE
XORLW k	立即数和 w 异或	Z

注:k 为立即数或标号(8 至 9 位的立即数值)。w 为工作寄存器(即累加器)。b 为位数,表是一个字节的段位。

d:若 d=0 或 d=w,操作结果放入 w 寄存器,若 d=1 或 d=f,则操作结果放入文件寄存器 f 中。对于立即数和控制类指令,k 表示 8 或 9 位的内容,或者表示立即数的值。

## 12.3 其他型号单片机及其生产厂商简介

(1) ATMEL 公司的 AVR 单片机,是增强型 RISC 内载 Flash 的单片机,芯片上的 Flash 存储器附在用户的产品中,可随时编程、再编程,使用户的产品设计容易,更新换代方便。AVR 单片机采用增强的 RISC 结构,使其具有高速处理能力,在一个时钟周期内可执行复杂的指令,每兆赫兹可实现 1 MIPS 的处理能力。AVR 单片机工作电压为 2.7~6.0 V,可以实现耗电最优化。AVR 的单片机广泛应用于计算机外部设备、工业实时控制、仪器仪表、通信设备、家用电器、宇航设备等各个领域。

(2) Motorola 单片机。Motorola 是世界上最大的单片机厂商。从 M6800 开始,开发了 4 位、8 位、16 位、32 位单片机,其中典型的代表有:8 位机 M6805,M68HC05 系列,8 位增强型 M68HC11,M68HC12,16 位机 M68HC16,32 位机 M683××。Motorola 单片机的特点之一是在同样的速度下所用的时钟频率较 Intel 类单片机低得多,因而使得高频噪声低,抗干扰能力强,更适合于工控领域及恶劣的环境。

(3) MicroChip 单片机。MicroChip 单片机的主要产品是 PIC16C 系列和 17C 系列 8 位单片机,CPU 采用 RISC 结构,分别仅有 33、35、58 条指令,采用 Harvard 双总线结构,运行速度快,低工作电压,低功耗,较大的输入输出直接驱动能力,价格低,一次性编程,体积小,适用于用量大,档次低,价格敏感的产品。在办公自动化设备、消费电子产品、电讯

通信、智能仪器仪表、汽车电子、金融电子、工业控制不同领域都有广泛的应用,PIC 系列单片机在世界单片机市场份额排名中逐年提高,发展非常迅速。

(4) MDT20××系列单片机,工业级 OTP 单片机,Micon 公司生产,与 PIC 单片机的管脚完全一致,海尔集团的电冰箱控制器,TCL 通信产品,长安、奥拓、铃木小轿车功率分配器就采用这种单片机。

(5) Scenix 单片机。Scenix 公司推出的 8 位 RISC 结构 SX 系列单片机与 Intel 的 Pentium II 等一起被评选为 1998 年世界十大处理器。在技术上有其独到之处: SX 系列双时钟设置,指令运行速度可达 50/75/100 MIPS(每秒执行百万条指令,××× M Instruction Per Second);具有虚拟外设功能,柔性化 I/O 端口,所有的 I/O 端口都可单独编程设定,公司提供各种 I/O 的库函数,用于实现各种 I/O 模块的功能,如多路 UART,多路 A/D, PWM, SPI, DTMF, FS, LCD 驱动等。采用 EEPROM/Flash 程序存储器,可以实现在线系统编程。通过计算机 RS232C 接口,采用专用串行电缆即可对目标系统进行在线实时仿真。

(6) EPSON 单片机。EPSON 单片机以低电压、低功耗和内置 LCD 驱动器特点闻名于世,尤其是 LCD 驱动部分做得很好,广泛用于工业控制、医疗设备、家用电器、仪器仪表、通信设备和手持式消费类产品等领域。目前 EPSON 已推出 4 位单片机 SMC62 系列、SMC63 系列、SMC60 系列和 8 位单片机 SMC88 系列。

(7) 东芝单片机。东芝单片机门类齐全,4 位机在家电领域有很大市场,8 位机主要有 870 系列、90 系列,该类单片机允许使用慢模式,采用 32 K 时钟时功耗降至 10 V·A 数量级。东芝的 32 位单片机采用 MIPS 3000A RISC 的 CPU 结构,面向 VCD、数字相机、图像处理等市场。

(8) 8051 单片机。8051 单片机最早由 Intel 公司推出,其后,多家公司购买了 8051 的内核,使得以 8051 为内核的 MCU 系列单片机在世界上产量最大,应用也最广泛,有人推测 8051 可能最终形成事实上的标准 MCU 芯片。

(9) GMS90 系列单片机。LG 公司生产的 GMS90 系列单片机,与 Intel MCS-51 系列,Atmel 89C2051、89C51/52, Atmel 89S51/52 等单片机兼容,CMOS 技术,高达 40 MHz 的时钟频率,应用于:多功能电话,智能传感器,电度表,工业控制,防盗报警装置,各种计费器,各种 IC 卡装置,DVD, VCD, CD-ROM。

(10) 华邦单片机。华邦公司的 W77、W78 系列 8 位单片机的引脚和指令集与 8051 兼容,但每个指令周期只需要 4 个时钟周期,速度提高了 3 倍,工作频率最高可达 40 MHz,同时增加了看门狗定时器,6 组外部中断源,两组 UART,双数据指针及等待状态控制引脚。W741 系列的 4 位单片机带液晶驱动,在线烧录,保密性高,低操作电压(1.2~1.8 V)。

(11) Zilog 单片机。Z8 单片机是 Zilog 公司的产品,采用多累加器结构,有较强的中断处理能力,开发工具价廉物美。Z8 单片机以低价位面向低端应用。很多人都知道 Z80 单片机,直到 20 世纪 90 年代后期,很多大学的微机原理还是讲述 Z80。

(12) NS 单片机。COP8 单片机是 NS(美国国家半导体公司)的产品,内部集成了 16

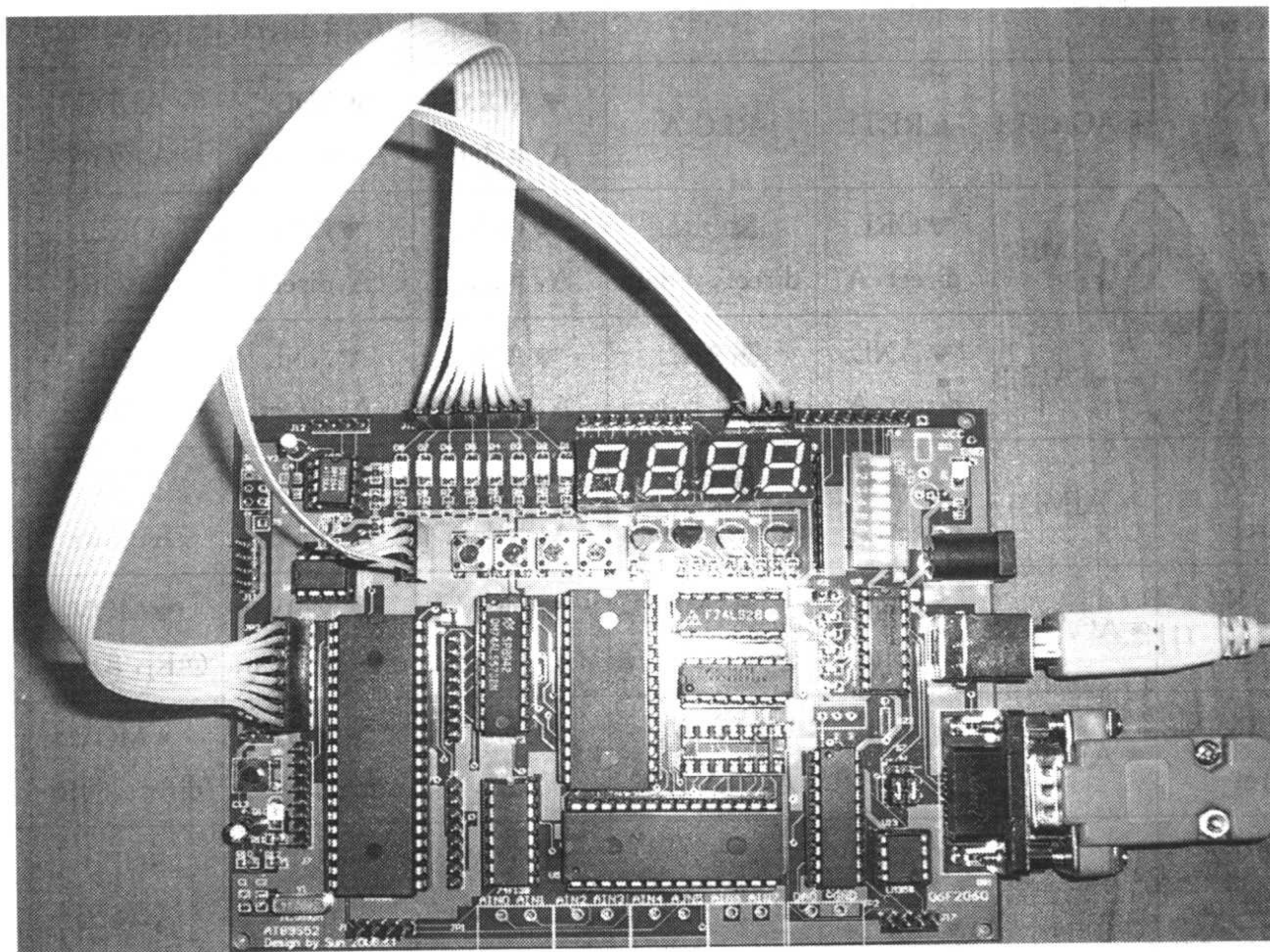
位A/ D,这是不多见的,在看门狗多路及 STOP 方式下单片机的唤醒方式上都有独到之处。此外,COP8 的程序加密也做得比较好。

(13) X1001 单片机。由珠海建荣科技公司推出,它是 100 MIPS 高性能 8 位 RISC 微控制器,采用了先进的 CMOS OTP 技术,应用于税控机、条形码扫描仪、打印机、网络控制系统、读卡器等。该款芯片性价比很高。



## 附录 AT89S52 单片机实验系统

本书配套实验板实物如附图 1 所示。可以完成书中所有的实践训练项目,其主要组成如下。



附图 1 实验板实物图

- CPU 为 Atmel 公司的 AT89S52,可利用 8751/8752,78E58/516 等替换。
- 晶振频率为 11.059 2 MHz,每个机器周期 1.25  $\mu$ s。
- 基本输入输出电路部分包括,8 位发光二极管,8 位拨码开关,4 个轻触开关。
- 显示部分为 4 个 7 段数码管(共阴极)。
- 扩展 8 KB 数据存储器 6264,串行存储器 24C256。
- 扩展 4 路 12 位 A/D(TLC0834)采样电路。
- 扩展 8 位 8 路 A/D(AD0809)采样电路。采样范围:0~5 V。
- 提供 1 路 8 位 D/A(DAC0832)转换电路,输出电压 0~3 V。
- 实用日历时钟芯片 DS1302,有年、月、日、分、秒等运行功能。
- 提供标准 RS232 标准串口,可与其他微机串口通讯,并兼作程序下载接口。
- 供电,外接 5 V 直流电源或计算机 USB 口。
- 数字温度测量芯片 DS18B20 接口。



附表1 AT89S52 指令速查表

低 高	0	1	2	3	4	5	6,7	8~F
0	NOP	* AJMP0	■LJMP addr16	RR A	INC A	▼INC direct	INC @Rj	INC Ri
1	■JBC bit, rel	* ACALL0	■LCALL addr16	RRC A	DECA	▼DEC direct	DEC @Ri	DRC Ri
2	■JB bit, rel	* AJMP1	▲RET	RLA	▼ADD A, # data	▼ADD A, # direct	ADD A, @Rj	ADD A, Ri
3	■JNB bit, rel	* ACALL1	▲RETI	RLC A	▼ADDC A, # data	▼ADDC A, # direct	ADDC A, @Rj	ADDC A, Ri
4	* JC rel	* AJMP2	▼ORL direct, A	■ORL direct, # data	▼ORL A, # data	▼ORL A, direct	ORL A, @Rj	ORL A, Ri
5	* JNC rel	* ACALL2	▼ANL direct, A	■ANL direct, # data	▼ANL A, # data	▼ANL A, direct	ANL A, @Rj	ANL A, Ri
6	* JZ rel	* AJMP3	▼XRL direct, A	■XRL direct, # data	▼XRL A, # data	▼XRL A, direct	XRL A, @Rj	XRL A, Ri
7	* JNZ rel	* ACALL3	* ORL c, bit	▲JMP @A+DPTR	▼MOV A, # data	■MOV direct, # data	▼MOV @Rj, # data	▼MOV Ri, # data
8	* SJMP rel	* AJMP 4	* ANL C, bit	▲MOVC A, @A+PC	DIV+ AB	■MOV direct, direct	* MOV direct, @Rj	* MOV direct, Ri
9	■MOVDRT R, # data	* ACALL4	▼MOV bit, C	▲MOVC A, @A+DPTR	▼SUBB A, # data	▼SUBB A, direct	SUBB A, @Rj	SUBB A, Ri
A	* ORL C, /bit	* AJMP 5	▼MOV bit, C	▲INC DPTR	+MUL AB		* MOV @Rj, direct	* MOV Ri, direct
B	* ANL C, /bit	* ACALL5	▼CPL bit	CPL C	+DIV AB	■CJNE A, direct, rel	■CJNE @Rj, # data, rel	■CJNE Ri, # data, rel
C	* PUSH direct	* AJMP 6	▼CLR bit	CLR C	SWAPA	▼XCH A, direct	XCH A, @Rj	XCH A, Ri
D	* POP direct	* ACALL6	▼SETB bit	SETB C	DA A	■DJNZ direct, rel	XCHD A, @Rj	* DJNZ Ri, rel
E	▲MOVX A, @DPTR	* AJMP 7	▲MOVX A, @R0	▲MOVX A, @R1	CLR A	▼MOV A, direct	MOV A, @Rj	MOV A, Ri
F	▲MOVX @DRTR, A	* ACALL7	▲MOVX @R0, A	▲MOVX @R1, A	CPL A	▼MOV direct, A	MOV @Rj, A	MOV Ri, A

注: ▲为单字节双周期指令。\* 双字节双周期指令。■为三字节双周期指令。+为单字节4周期指令。▼为双字节单周期指令。其他均为单字节单周期指令。

附表 2 AT89S52 指令详表

指令名称	指令代码	字节数	机器周期
ACALL addr11	A10A9A810001A7A6A5A4A3A2A1A0	2	2
ADD A, Rn	28H~2FH	1	1
ADD A, direct	25H	2	1
ADD A, @Ri	26H~27H	1	1
ADD A, # data	24H	2	1
ADDC A, Rn	38H~3FH	1	1
ADDC A, direct	35H	2	1
ADDC A, @Ri	36H~37H	1	1
ADDC A, # data	34H	2	1
AJMP addr11	A10A9A800001A7A6A5A4A3A2A1A0	2	2
ANL A, Rn	58H~5FH	1	1
ANL A, direct	55H	2	1
ANL A, @Ri	56H~57H	1	1
ANL A, # data	54H	2	1
ANL direct, A	52H	2	1
ANL direct, # data	53H	3	2
ANL C, bit	82H	2	2
ANL C, /bit	B0H	2	2
CJNE A, direct, rel	B5H	3	2
CJNE A, # data, rel	B4H	3	2
CJNE Rn, # data, rel	B8H~BFH	3	2
CJNE @Ri, # data, rel	B6H~B7H	3	2
CLR A	E4H	1	1
CLR C	C3H	1	1
CLR bit	C2H	2	1
CPL A	F4H	1	1
CPL C	B3H	1	1
CPL bit	B2H	2	1
DA A	D4H	1	1
DEC A	14H	1	1
DEC Rn	18H~1FH	1	1



续表

指令名称	指令代码	字节数	机器周期
DEC direct	15H	2	1
DEC @Ri	16H~17H	1	1
DIV AB	84H	1	4
DJNZ Rn, rel	D8H~DFH	2	2
DJNZ direct, rel	D5H	3	2
INC A	04H	1	1
INC Rn	08H~0FH	1	1
INC direct	05H	2	1
INC @Ri	06H~07H	1	1
INC DPTR	A3H	1	2
JB bit, rel	20H	3	2
JBC bit, rel	10H	3	2
JC rel	40H	2	2
JMP @A+DPTR	72H	1	2
JNB bit, rel	30H	3	2
JNC rel	50H	2	2
JNZ rel	70H	2	2
JZ rel	60H	2	2
LCALL addr16	12H	3	2
LJMP addr16	02H	3	2
MOV A, Rn	E8H~EFH	1	1
MOV A, direct	E5H	2	1
MOV A, @Ri	E6H~E7H	1	1
MOV A, #data	74H	2	1
MOV Rn, A	F8H~FFH	1	1
MOV Rn, direct	A8H~AFH	2	2
MOV Rn, #data	78H~7FH	2	1
MOV direct, A	F5H	2	1
MOV direct, Rn	88H~8FH	2	2
MOV direct2, direct1	85H	3	2
MOV direct, @Ri	86H~87H	2	2

续 表

指令名称	指令代码	字节数	机器周期
MOV direct, # data	75H	3	2
MOV @Ri, A	F6H~F7H	1	1
MOV @Ri, direct	A6H~A7H	2	2
MOV @Ri, data	76H~77H	2	1
MOV C, bit	A2H	2	1
MOV bit, C	92H	2	2
MOV DPTR, # data16	90H	3	2
MOVC A, @A+DPTR	93H	1	2
MOVC A, @A+PC	83H	1	2
MOVX A, @Ri	E2H~E3H	1	2
MOVX A, @DPTR	E0H	1	2
MOVX @Ri, A	F2H~F3H	1	2
MOVX @DPTR, A	F0H	1	2
MUL AB	A4H	1	4
NOP	00H	1	1
ORL A, Rn	48H~4FH	1	1
ORL A, direct	45H	2	1
ORL A, @Ri	46H~47H	1	1
ORL A, # data	44H	2	1
ORL direct, A	42H	2	1
ORL direct, # data	43H	3	2
ORL C, bit	72H	2	2
ORL C, /bit	A0H	2	2
POP direct	D0H	2	2
PUSH direct	C0H	2	2
RET	22H	1	2
RETI	32H	1	2
RL A	23H	1	1
RLC A	33H	1	1
RR A	03H	1	1
RRC A	13H	1	1

续表

指令名称	指令代码	字节数	机器周期
SETB C	D3H	1	1
SETB bit	D2H	2	1
SJMP rel	80H	2	2
SUBB A, Rn	98H~9FH	1	1
SUBB A, direct	95H	2	1
SUBB A, @Ri	96H~97H	1	1
SUBB A, # data	94H	2	1
SWAP A	C4H	1	1
XCH A, Rn	C8H~CFH	1	1
XCH A, direct	C5H	2	1
XCH A, @Ri	C6H~C7H	1	1
XCHD A, @Ri	D6H~D7H	1	1
XRL A, Rn	68H~6FH	1	1
XRL A, direct	65H	2	1
XRL A, @Ri	66H~67H	1	1
XRL A, # data	64H	2	1
XRL direct, A	62H	2	1
XRL direct, # data	63H	3	2

附表3 C51 中的关键字

关键字	用途	说明
auto	存储种类说明	用以说明局部变量, 默认值为此
break	程序语句	退出最内层循环
case	程序语句	switch 语句中的选择项
char	数据类型说明	单字节整型数或字符型数据
const	存储类型说明	在程序执行过程中不可更改的常量值
continue	程序语句	转向下一次循环
default	程序语句	switch 语句中的失败选择项
do	程序语句	构成 do...while 循环结构
double	数据类型说明	双精度浮点数
else	程序语句	构成 if...else 选择结构
enum	数据类型说明	枚举
extern	存储种类说明	在其他程序模块中说明了的全局变量

续表

关键字	用途	说明
float	数据类型说明	单精度浮点数
for	程序语句	构成 for 循环结构
goto	程序语句	构成 goto 转移结构
if	程序语句	构成 if...else 选择结构
int	数据类型说明	基本整型数
long	数据类型说明	长整型数
register	存储种类说明	使用 CPU 内部寄存的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数, 二进制数据的最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	结构类型数据
switch	程序语句	构成 switch 选择结构
typedef	数据类型说明	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数数据
void	数据类型说明	无类型数据
volatile	数据类型说明	该变量在程序执行中可被隐含地改变
while	程序语句	构成 while 和 do...while 循环结构

附表 4 ANSIC 标准关键字

关键字	用途	说明
bit	位标量声明	声明一个位标量或位类型的函数
sbit	位标量声明	声明一个可位寻址变量
Sfr	特殊功能寄存器声明	声明一个特殊功能寄存器
Sfr16	特殊功能寄存器声明	声明一个 16 位的特殊功能寄存器
data	存储器类型说明	直接寻址的内部数据存储器
bdata	存储器类型说明	可位寻址的内部数据存储器
idata	存储器类型说明	间接寻址的内部数据存储器
pdata	存储器类型说明	分页寻址的外部数据存储器
xdata	存储器类型说明	外部数据存储器
code	存储器类型说明	程序存储器
interrupt	中断函数说明	定义一个中断函数
reentrant	再入函数说明	定义一个再入函数
using	寄存器组定义	定义芯片的工作寄存器

附表 5 AT89S52 特殊功能寄存器

序号	符 号	地 址	注 释
1	* ACC	0E0H	累加器
2	* B	0F0H	乘法寄存器
3	* PSW	0D0H	程序状态字
4	SP	81H	堆栈指针
5	* IE	0A8H	中断允许控制器
6	* IP	0D8H	中断优先控制器
7	* P0	80H	端口 0
8	* P1	90H	端口 1
9	* P2	0A0H	端口 2
10	* P3	0B0H	端口 3
11	PCON	87H	电源控制及波特率选择
12	* SCON	98H	串行口控制器
13	SBUF	99H	串行数据缓冲器
14	* TCON	88H	定时器控制
15	TMOD	89H	定时器方式选择
16	TL0	8AH	定时器 0 低 8 位
17	TL1	8BH	定时器 1 低 8 位
18	TH0	8CH	定时器 0 高 8 位
19	TH1	8DH	定时器 1 高 8 位
20	DP0L	82H	数据指针 DPTR0 的低 8 位
21	DP0H	83H	数据指针 DPTR0 的高 8 位
22	DP1L	84H	数据指针 DPTR1 的低 8 位
23	DP1H	85H	数据指针 DPTR1 的高 8 位
24	TL2	0CCH	定时器 2 低 8 位
25	TH2	0CDH	定时器 2 高 8 位
26	T2CON *	0C8H	定时器 2 控制寄存器
27	T2MOD	0C9H	定时器 2 模式寄存器
28	RCAP2L	0CAH	定时器 2 捕捉/重装寄存器低 8 位
29	RCAP2H	0CBH	定时器 2 捕捉/重装寄存器高 8 位
30	AUX	08EH	辅助寄存器
31	AUXR1	0A2H	辅助寄存器 1
32	WDTRST	0A6H	WDT 复位寄存器

注:带 \* 号的特殊功能寄存器都是可以位寻址的寄存器。

附表 6 C 语言运算符优先级和结合性

级 别	类 别	名 称	运 算 符	结 合 性
1	强制转换、数组、 结构、联合	强制类型转换	( )	右结合
		下标	[ ]	
		存取结构或联合成员	->或.	
2	逻辑	逻辑非	!	左结合
	字 位	按位取反	~	
	增 量	加一	++	
	减 量	减一	--	
	指 针	取地址	&	
		取内容	*	
	算 术	单目减	-	
	长度计算	长度计算	sizeof	
3	算 术	乘	*	右结合
		除	/	
		取模	%	
4	算术和指针运算	加	+	
		减	-	
5	字 位	左移	<<	
		右移	>>	
6	关 系	大于等于	>=	
		大于	>	
		小于等于	<=	
		小于	<	
		恒等于	==	
7		不等于	!=	
8	字 位	按位与	&	
9		按位异或	^	
10		按位或		
11	逻辑	逻辑与	&&	左结合
12		逻辑或		
13	条 件	条件运算	?:	
14	赋 值	赋值	=	
		复合赋值	Op=	
15	逗 号	逗号运算	,	右结合

附表7 ASCII码表

DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX	CHR
0	00	NUL	43	2B	+	86	56	V
1	01	SOH	44	2C	,	87	57	W
2	02	STX	45	2D	-	88	58	X
3	03	ETX	46	2E	.	89	59	Y
4	04	EOT	47	2F	/	90	5A	Z
5	05	ENQ	48	30	0	91	5B	[
6	06	ACK	49	31	1	92	5C	\
7	07	BEL	50	32	2	93	5D	]
8	08	BS	51	33	3	94	5E	^
9	09	TAB	52	34	4	95	5F	_
10	0A	LF	53	35	5	96	60	`
11	0B	VT	54	36	6	97	61	a
12	0C	FF	55	37	7	98	62	b
13	0D	CR	56	38	8	99	63	c
14	0E	SO	57	39	9	100	64	d
15	0F	SI	58	3A	:	101	65	e
16	10	DLE	59	3B	;	102	66	f
17	11	DC1	60	3C	<	103	67	g
18	12	DC2	61	3D	=	104	68	h
19	13	DC3	62	3E	>	105	69	i
20	14	DC4	63	3F	?	106	6A	j
21	15	NAK	64	40	@	107	6B	k
22	16	SYN	65	41	A	108	6C	l
23	17	ETB	66	42	B	109	6D	m
24	18	CAN	67	43	C	110	6E	n
25	19	EM	68	44	D	111	6F	o
26	1A	SUB	69	45	E	112	70	p
27	1B	ESC	70	46	F	113	71	q
28	1C	FS	71	47	G	114	72	r
29	1D	GS	72	48	H	115	73	s
30	1E	RS	73	49	I	116	74	t
31	1F	US	74	4A	J	117	75	u
32	20	(space)	75	4B	K	118	76	v
33	21	!	76	4C	L	119	77	w
34	22	"	77	4D	M	120	78	x
35	23	#	78	4E	N	121	79	y
36	24	\$	79	4F	O	122	7A	z
37	25	%	80	50	P	123	7B	{
38	26	&	81	51	Q	124	7C	
39	27	'	82	52	R	125	7D	}
40	28	(	83	53	S	126	7E	~
41	29	)	84	54	T	127	7F	DEL
42	2A	*	85	55	U			



附表 8 主要单片机生产厂商网址及相关信息网址

美国国家半导体公司	<a href="http://www.national.com">http://www.national.com</a>
Zilog 公司	<a href="http://www.zilog.com">http://www.zilog.com</a>
Atmel 公司	<a href="http://www.atmel.com">http://www.atmel.com</a>
Motorola 公司	<a href="http://www.mot.com/">http://www.mot.com/</a>
MicroChip 公司	<a href="http://www.microchip.com/">http://www.microchip.com/</a>
Scenix 公司	<a href="http://www.scenix.com">http://www.scenix.com</a>
EPSON 公司	<a href="http://www.epson.com">http://www.epson.com</a>
LG 公司	<a href="http://www.lgs.co.kr">http://www.lgs.co.kr</a>
三星公司	<a href="http://www.samsungsemi.com">http://www.samsungsemi.com</a>
华邦公司	<a href="http://www.winbond.com.tw">http://www.winbond.com.tw</a>
建荣科技公司	<a href="http://www.buildwin.com.cn/">http://www.buildwin.com.cn/</a>
广州周立功单片机发展有限公司	<a href="http://www.zlgmcu.com">http://www.zlgmcu.com</a>
武汉力源信息技术有限公司	<a href="http://www.icbase.com">http://www.icbase.com</a>
万利电子(南京)有限公司	<a href="http://www.manley.com.cn">http://www.manley.com.cn</a>
南京西尔特电子有限公司	<a href="http://www.xeltek-cn.com">http://www.xeltek-cn.com</a>

## 参 考 文 献

- [1] 张淑清,等. 单片微型计算机接口技术及其应用. 北京:国防工业出版社,2001.
- [2] 李朝青. 单片机原理及接口技术. 北京:北京航空航天大学出版社,2003.
- [3] 孙育才,等. AT89S52 系列单片机原理及其应用. 北京:清华大学出版社,2004.
- [4] 何立民. MCS-51 系列单片机应用系统设计. 北京:北京航空航天大学出版社,1990.
- [5] 马忠梅,等. 单片机的 C 语言应用程序设计. 北京:北京航空航天大学出版社,1998.
- [6] 李华,等. MCS-51 系列单片机实用接口技术. 北京:北京航空航天大学出版社,2002.
- [7] 李群芳,等. 单片微型计算机与接口技术. 北京:电子工业出版社,2001.
- [8] 余锡存,等. 单片机原理及接口技术. 西安:西安电子科技大学出版社,1999.
- [9] 徐安,等. 单片机原理与应用. 北京:北京希望电子出版社,2003.
- [10] 杨恢先,等. 单片机原理及应用. 北京:国防科技大学出版社,2003.
- [11] 王幸之,等. AT89 系列单片机原理与接口技术. 北京:北京航空航天大学出版社,2004.
- [12] 林伸茂,等. 8051 单片机彻底研究经验篇. 北京:人民邮电出版社,2004.
- [13] AT89S52, 2001.
- [14] 周立功,等. 增强型 80C51 单片机速成与实战. 北京:北京航空航天大学出版社,2003.
- [15] 李伯成. 基于 MCS-51 单片机的嵌入式系统设计. 北京:电子工业出版社,2004.
- [16] Xicor: Xicor Application Note, Interfacing the X84041 to 8051 Microcontrollers, 1996.
- [17] Xicor: X84041 Micro Port Saver E<sup>2</sup>PROM, 1996.
- [18] TEXAS Instruments: TLV5616C, TLV5616I 2.7 V to 5.5 V Low Power 12-Bit Digital-To-Analog Converters with Power down, 1995.
- [19] TEXAS Instruments: Microcontroller Based Data Acquisition Using report the TLC2543 12-Bit Serial-out ADC Application, 1995.
- [20] Dallas: Application Note 82 Using the Dallas Trickle Charge Timekeeper, 1996.